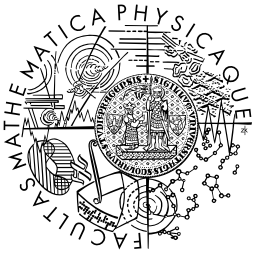


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁRSKA PRÁCA

Adam Szabó

Využitie branch and bound prístupu pre parametrické intervalové lineárne sústavy

Katedra aplikované matematiky

Vedúci bakalárskej práce:	RNDr. Jaroslav Horáček
Študijný program:	Informatika
Študijný obor:	Všeobecná informatika

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Rád by som poďakoval vedúcemu práce RNDr. Jaroslavovi Horáčkovi za jeho cenné rady, ochotu pri konzultáciách, odbornú pomoc a predovšetkým za jeho ľudský prístup pri vypracovaní tejto bakalárskej práce. Osobitné poďakovanie patrí mojej rodine a priateľom za podporu a pochopenie.

Název práce: Využití branch and bound přístupu pro parametrické intervalové lineární soustavy

Autor: Adam Szabó

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Jaroslav Horáček, Katedra aplikované matematiky

Abstrakt: Tato práce se zaobírá parametrickými intervalovými lineárními soustavami. Branch and bound metodou a různými námi implementovanými prořezávacími podmínkami jsme dostali jejich množinu řešení. Přesněji jsme ji popsali pomocí n-rozměrných boxů, které jsme získali díky využitým metodám. Seznámili jsme se se základními pojmy ohledně intervalů a lineárních soustav. Následně jsme zpracovávali získané boxy řešení vícerými metodami s cílem optimalizovat jejich počet. Součástí práce je i porovnání jednotlivých prořezávacích podmínek na parametrických soustavách s různým počtem parametrů. Uvedené algoritmy byly implementované do intervalového balíku Lime s možností jednoduché vizualizace získaných řešení.

Klíčová slova: intervalová analýza, parametrické intervalové linearní soustavy, obálka množiny řešení, branch and bound

Title: Application of Branch and Bound Approach to Parametric Interval Linear Systems

Author: Adam Szabó

Department: Department of Applied Mathematics

Supervisor: RNDr. Jaroslav Horáček, Department of Applied Mathematics

Abstract: This work is focused on parametric interval linear systems. By using branch and bound method and various pruning conditions, we first obtained their solution and then described it more precisely with n-dimensional boxes. We were acquainted with the basic concepts of intervals and linear systems. Subsequently, we processed the boxes obtained by multiple methods to optimize their number. Part of the work is also a comparison of various pruning conditions on parametric systems with the different number of parameters. Finally, our algorithms were implemented into the Lime interval package with the possibility of simple visualization of the obtained solutions.

Keywords: interval analysis, parametric interval linear systems, enclosure of a solution set, branch and bound

Názov práce: Využitie branch and bound prístupu pre parametrické intervalové lineárne sústavy

Autor: Adam Szabó

Katedra: Katedra aplikované matematiky

Vedúci bakalárskej práce: RNDr. Jaroslav Horáček, Katedra aplikované matematiky

Abstrakt: Táto práca sa zaoberá parametrickými intervalovými lineárnymi sústavami. Branch and bound metódou a rôznymi nami implementovanými prerezávacími podmienkami sme dostali ich množinu riešení. Presnejšie sme ju popísali pomocou n-rozmerných boxov, ktoré sme získali vďaka využitým metódam. Zoznámili sme sa so základnými pojmami ohľadne intervalov a lineárnych sústav. Následne sme spracovávali získané boxy riešení viacerými metódami s cieľom optimalizovať ich počet. Súčasťou práce je aj porovnanie jednotlivých prerezávacích podmienok na parametrických sústavách s rôznym počtom parametrov. Uvedené algoritmy boli implementované do intervalového balíku Lime s možnosťou jednoduchej vizualizácie získaných riešení.

Kľúčové slová: intervalová analýza, parametrické intervalové lineárne sústavy, obálka množiny riešení, branch and bound

Obsah

Úvod

Jeden zo základných pojmov potrebných v našej práci bude lineárna sústava. Je to dôležitý pojem, pretože všeobecne vieme väčšinu problémov previesť na riešenie lineárnych sústav, napr. derivácie, integrácie, atď. Súbor lineárnych rovníc v sebe obsahuje len samotné čísla, ale v praxi sa stretávame s rôznymi nepresnosťami ako sú napríklad zaokrúhľovacie chyby a preto nahrádzame čísla intervalmi. Takto nám vzniknú intervalové lineárne sústavy, teda súbor lineárnych rovníc, ktoré majú ako koeficient reálne uzavreté intervaly.

Väčšina problémov s technickým riešením ako modely v operačnom výskume alebo lineárne predikčné problémy a pod., obsahuje zvyčajne koeficienty, ktoré na sebe závisia. Hlavným dôvodom tejto závislosti je, že chyby v niekoľkých rôznych koeficientoch môžu byť spôsobené rovnakým faktorom. Jeden parameter sa môže premietať do viacerých koeficientov a tým pádom získavame tzv. parametrickú intervalovú lineárnu sústavu. Množina riešení takýchto sústav má všeobecne nekonvexný nelineárny charakter.

Lineárne závislosti skúmali viacerí autori. Prvý článok o parametrických intervalových systémoch vypracoval Jansson [9]. Všeobecný problém lineárnych systémov závislých od intervalových parametrov bol prvý krát spracovaný Rumpom v roku 1994 [24]. Pre špecifickú triedu parametrických systémov navrhli Neumaier a Pownuk [16] efektívnu metódu. Riešenie parametrických lineárnych systémov je v dnešnej dobe čoraz dôležitejšie, keďže zahŕňa neurčitosť parametrov. Je dôležitou súčasťou riešení mnohých vedeckých a technických problémov. Nech sa jedná o počítanie s hodnotami, ktoré sú merané s určitou nepresnosťou alebo počítanie zaokrúhľovacích chýb pri numerických výpočtoch na počítači.

Vo väčšine prác s tématikou parametrických intervalových lineárnych sústav sa autori snažia riešenie zapúzdriť jedným boxom. To je však často nevýhodné, pretože riešenie spomínaných sústav môže mať zvláštny tvar a je zbytočne nadhodnotené len jedným boxom. Z tohoto dôvodu je výhodnejšie pracovať s viacerými boxmi, na čo využívame metódu branch and bound. Metóda branch and bound alebo inými slovami metóda vetiev a hraníc bola prvý krát navrhnutá dvomi pánmi Landom a Doigom v roku 1960 [12] pre lineárne programovanie. V podstate je to všeobecný algoritmus v lineárnom programovaní pre hľadanie optimálnych riešení rôznych optimalizačných problémov pomocou obmedzeného prehľadávania.

Cieľom tejto práce bude popísať množiny riešení pomocou n-rozmerných boxov tak, aby sme dostali čo najpresnejší popis. Výpočet intervalového obalu týchto množín je vo všeobecnosti NP-ťažký problém rovnako ako rozhodnúť, či je táto množina prázdna alebo neprázdna. Pre zapúzdrenie množiny n-rozmerným boxom existuje hneď niekoľko metód, tie však môžu byť príliš hrubým popisom vo zväčša komplikovanom tvare množiny riešení. V tejto práci sa budeme snažiť získať spomínané n-rozmerné boxy metódou branch and bound v kombinácii s vhodnými prerezávacími podmienkami. Následne sa budeme zaoberať spracovaním získaných boxov (vhodné zlievanie, minimalizácia ich počtu a pod.) a ich vizualizáciou. Ďalším cieľom je implementovanie popísanej metódy do intervalového balíku Lime.

1. Základné pojmy

V tejto kapitole najprv uvedieme základné pojmy a označenia najmä ohľadne intervalov [21]. Ďalej si ukážeme ako sa s intervalmi pracuje. Väčšina základných definícií je prevzatá z diplomovej práce J. Horáčka [8] a z knihy Moore a kol.[13].

1.1 Interval a súvisiace pojmy

Definícia 1 (Reálny interval)**.** *Nech* $\underline{a}, \overline{a} \in \mathbb{R}$ *a nech platí* $\underline{a} \leq \overline{a}$*. Potom množinu*

$$\boldsymbol{a} = [\underline{a}, \overline{a}] := \{a \in \mathbb{R} \mid \underline{a} \leq a \leq \overline{a}\}$$

nazývame reálnym intervalom. Číslo $\underline{a}$*, resp.* $\overline{a}$ *nazývame dolnou, resp. hornou medzou. Množinu všetkých reálnych intervalov budeme označovať symbolom* $\mathbb{IR}$*.*

Definícia 2. *Nech* $\boldsymbol{a} \in \mathbb{IR}$*. Ak platí* $\underline{a} = \overline{a}$*, potom povieme, že interval je degenerovaný.*

Definícia 3. *Nech* $\boldsymbol{a} \in \mathbb{IR}$*. Ak platí* $\underline{a} = -\overline{a}$*, potom povieme, že interval je symetrický.*

Definícia 4 (Intervalové pojmy)**.** *Nech* $\boldsymbol{a} \in \mathbb{IR}$*, potom definujeme pojmy:*

1. *šírka intervalu* $\boldsymbol{a}$*:* $\mathrm{w}(\boldsymbol{a}) = \overline{a} - \underline{a}$*,*
2. *stred intervalu* $\boldsymbol{a}$*:* $a^c = \frac{1}{2}(\underline{a} + \overline{a})$*,*
3. *polomer intervalu* $\boldsymbol{a}$*:* $a^{\triangle} = \frac{1}{2}(\overline{a} - \underline{a})$*,*
4. *mignitúda intervalu* $\boldsymbol{a}$*:* $\mathrm{mig}(\boldsymbol{a}) = \min(|\underline{a}|, |\overline{a}|)$*,*
5. *magnitúda intervalu* $\boldsymbol{a}$*:* $\mathrm{mag}(\boldsymbol{a}) = \max(|\underline{a}|, |\overline{a}|)$*,*
6. *absolútna hodnota intervalu* $\boldsymbol{a}$*:* $|\boldsymbol{a}| = \{|a|; a \in \boldsymbol{a}\}$*.*

Množinu všetkých n-zložkových intervalových vektorov budeme označovať $\mathbb{IR}^n$. Intervalové pojmy pre nich platia po zložkách.

Definícia 5 (Intervalová matica)**.** *Nech sú* $\underline{A}, \overline{A} \in \mathbb{R}^{m \times n}$ *a platí (po zložkách) nerovnosť* $\underline{A} \leq \overline{A}$*, potom intervalovú maticu definujeme ako*

$$\boldsymbol{A} = [\underline{A}, \overline{A}] = \{\forall A \in \boldsymbol{A};\ \underline{A} \leq A \leq \overline{A}\}.$$

Maticu $\underline{A}$ *nazývame dolnou medzou a maticu* $\overline{A}$ *hornou medzou.*

Intervalové matice a vektory si označme tučne: $\mathbf{A}$ a $\mathbf{b}$. Majme všetky možné prvky intervalových koeficientov matice $\mathbf{A}$. Potom sú to práve koeficienty množiny všetkých matíc, čo je vlastne intervalová matica $\mathbf{A}$. Je dôležité, aby sme vyberali prvky z daných intervalov nezávisle. Vidíme, že v prípade rovnosti $\underline{A} = \overline{A}$ nám vychádza klasická neintervalová matica.

V nasledujúcom kroku si zadefinujeme stredovú maticu a maticu polomeru. Keďže už vieme, čo je dolná a horná medza intervalovej matice, tak si už ľahko vieme odvodiť, že platí:

$$A^c = \frac{1}{2}(\underline{A} + \overline{A}),$$

$$A^\triangle = \frac{1}{2}(\overline{A} - \underline{A}).$$

Znalosť stredovej matice A^c a matice polomeru $A^\triangle$ nám umožňuje vyjadriť už vyššie definovanú intervalovú maticu alternatívnym spôsobom ako

$$\mathbf{A} = \{\forall A \in \mathbf{A};\ |A - A^c| \leq A^\triangle\}.$$

Taktiež je možné z týchto matíc spätne odvodiť ako vyzerá horná a dolná medza a to nasledovne:

$$\underline{A} = A^c - A^\triangle,$$

$$\overline{A} = A^c + A^\triangle.$$

Definícia 6. *Štvorcová intervalová matica $\boldsymbol{A}$ sa nazýva regulárna, ak pre každú reálnu maticu $A \in \boldsymbol{A}$ platí, že je regulárna.*

Definícia 7. *Štvorcová intervalová matica $\boldsymbol{A}$ sa nazýva singulárna, ak existuje reálna matica $A \in \boldsymbol{A}$, ktorá je singulárna.*

1.2 Množinové operácie

Intervaly sú vlastne množiny a preto je možné na ne aplikovať množinové operácie.

Definícia 8 (Množinové operácie)**.** *Nech $\boldsymbol{a} = [\underline{a}, \overline{a}], \boldsymbol{b} = [\underline{b}, \overline{b}] \in \mathbb{IR}$. Potom definujeme nasledujúce operácie.*

Prienik

$$\boldsymbol{a} \cap \boldsymbol{b} := \{c \mid c \in \boldsymbol{a} \wedge c \in \boldsymbol{b}\}.$$

Prienik $\boldsymbol{a},\ \boldsymbol{b}$ je prázdny, pokiaľ $\overline{b} \leq \underline{a}$ alebo $\overline{a} \leq \underline{b}$ a píšeme $\boldsymbol{a} \cap \boldsymbol{b} = \emptyset$. V prípade, že je neprázdny, ho môžeme ekvivalentne vyjadriť ako $\boldsymbol{a} \cap \boldsymbol{b} = [\max(\underline{a}, \underline{b}),\ \min(\overline{a}, \overline{b})]$.

Zjednotenie

$$\boldsymbol{a} \cup \boldsymbol{b} := \{c \mid c \in \boldsymbol{a} \vee c \in \boldsymbol{b}\}.$$

Zjednotenie $\boldsymbol{a} \cup \boldsymbol{b}$ môže byť vo všeobecnosti viac intervalov. Preto si zavedieme tzv. obal, ktorý je najmenším intervalom (vzhľadom k inklúzii) obsahujúcim množinu $\boldsymbol{a} \cup \boldsymbol{b}$.

Obal

$$\boldsymbol{a} \underline{\cup} \boldsymbol{b} := [\min(\underline{a}, \underline{b}),\ \max(\overline{a}, \overline{b})].$$

Ak by sme cheli definovať množinové operácie pre intervalové vektory, tak by sme to spravili po zložkách.

1.3 Porovnávanie intervalov

V nasledujúcom odstavci si ukážeme podobne ako v práci [4] ako je možné porovnávanie jednotlivých prvkov. Budeme potrebovať množinu reálnych intervalov, na ktorú dané porovnávanie zavedieme. Využijeme operácie $\leq$, $=$ na $\mathbb{R}$. Nech máme dva intervaly $\mathbf{a} = [\underline{a}, \overline{a}]$ a $\mathbf{b} = [\underline{b}, \overline{b}]$, ktoré sú reálne. Potom

1. sú si rovné práve vtedy, keď platí:

$$\mathbf{a} = \mathbf{b} \Leftrightarrow \underline{a} = \underline{b} \wedge \overline{a} = \overline{b}.$$

   Teda vidíme, že popisujú rovnakú množinu.

2. $\mathbf{a}$ je menší (nanajvýš rovný) $\mathbf{b}$, keď pre každú dvojicu prvkov (a,b) platí:

$$a \leq b, \quad \text{kde } a \in \mathbf{a},\ b \in \mathbf{b}.$$

   Teda môžeme jednoducho napísať, že $\mathbf{a} \leq \mathbf{b} \Leftrightarrow \overline{a} \leq \underline{b}$.

Podobným spôsobom definujeme operácie $\geq$, $>$, $<$.

Môžeme si všimnúť, že existujú neporovnateľné prvky. Dôsledkom toho je, že takto definované usporiadanie na reálnych intervaloch nie je úplne, narozdiel od operácie $\leq$ na $\mathbb{R}$, kde vektory a matice porovnávame po zložkách. Môžeme to vidieť na nasledujúcom príklade: majme intervaly [1, 2] a [1, 3]. Pre tieto intervaly neplatí $[1, 2] \leq [1, 3]$ a ani $[1, 3] \leq [1, 2]$.

1.4 Intervalová aritmetika

Aritmetické operácie na reálnych intervaloch sú vlastne rozšírenia známych operácií z $\mathbb{R}$, len namiesto čísla použijeme interval. Vďaka tomu máme voľnosť pre určité nepresnosti ako napríklad chyby meriania alebo zaokrúhľovacie chyby, čo je v bežnom používaní obrovská výhoda. Je však nutné dodržať určité pravidlá. Je teda nutné, aby výsledné intervaly zahŕňali hodnoty operácií pre všetky možné voľby prvkov z intervalov a taktiež požadujeme, aby sme po každej intervalovej operácii dostali znovu interval.

Definícia 9 (Aritmetické operácie)**.** *Majme intervaly* $\boldsymbol{a} \in \mathbb{IR}, \boldsymbol{b} \in \mathbb{IR}$. *Potom aritmetické operácie* $\circ$ *definujeme nasledovne:*

$$\boldsymbol{a} \circ \boldsymbol{b} := \{a \circ b \mid a \in \boldsymbol{a}, b \in \boldsymbol{b}\}.$$

Pre konkrétne operácie $+, -, *, /$ si odvodíme priamy výpočet.

$$\begin{aligned}
&\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}],\\
&\mathbf{a} - \mathbf{b} = [\underline{a} - \overline{b}, \overline{a} + \underline{b}],\\
&\mathbf{a} * \mathbf{b} = [\min(X),\ \max(X)], \quad \text{kde } X = \{\underline{a} * \underline{b}, \underline{a} * \overline{b}, \overline{a} * \underline{b}, \overline{a} * \overline{b}\},\\
&\mathbf{a}/\mathbf{b} = \mathbf{a} * (1/\mathbf{b}), \quad \text{kde } 1/\mathbf{b} = [1/\overline{b}, 1/\underline{b}],\ 0 \notin \mathbf{b}.
\end{aligned}$$

Je dôležité zmieniť, že nie všetky operácie v $\mathbb{R}$ platia aj v $\mathbb{IR}$. Množina $\mathbb{IR}(+, *, 0, 1)$ nie je teleso, ani len okruh, ale aj napriek tomu pre ňu niektoré

vlastnosti telesa platia. Napríklad tu existuje nulový prvok $\mathbf{0} \equiv [0, 0]$ a aj jednotkový prvok $\mathbf{1} \equiv [1, 1]$. Tieto prvky sa však nerovnajú. Pre každý interval x potom platí:

$$\mathbf{0} + \mathbf{a} = \mathbf{a},$$

$$\mathbf{1} * \mathbf{a} = \mathbf{a},$$

$$\mathbf{0} * \mathbf{a} = \mathbf{0}.$$

Vieme, že operácie násobenia a sčítania sú komutatívne a asociatívne, nie však distributívne. Dôsledkom toho závádzame slabšiu subdistributivitu:

$$\mathbf{a}(\mathbf{b} + \mathbf{c}) \subseteq \mathbf{ab} + \mathbf{ac}.$$

Taktiež všeobecne platí, že nedegenerované intervaly nemajú opačný (vzhľadom k sčítaniu) a inverzný (vzhľadom k násobeniu) prvok.

2. Sústavy

2.1 Lineárne sústavy

Definícia 10. *Sústavou m lineárnych rovníc o n neznámych rozumieme sústavu v tvare*

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m, \end{aligned}$$

kde a_{ij}, $b_i \in \mathbb{R}$, *pre* $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ *sú dané koeficienty a* $x_1, \ldots, x_n$ *sú neznáme.*

Zapisujeme ju taktiež v tvare

$$Ax = b,$$

kde A $\in \mathbb{R}^{m\times n}$ a x, $b \in \mathbb{R}^m$.

2.2 Intervalové lineárne sústavy

Intervalové lineárne sústavy rovníc sú sústavy, ktoré majú ako koeficienty reálne uzavreté intervaly.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ \vdots \quad\quad \vdots \quad\quad\quad\quad \vdots \quad\quad \vdots & \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m, \end{aligned}$$

kde $a_{ij} \in \boldsymbol{a} = [\underline{a}_{ij}, \overline{a}_{ij}]$, pre $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$, a $b_i \in \boldsymbol{b} = [\underline{b}_i, \overline{b}_i]$, pre $i \in \{1, \ldots, m\}$.

Definícia 11. *Nech* $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ *je intervalová matica a* $\boldsymbol{b} \in \mathbb{IR}^m$ *je intervalový vektor. Potom intervalovým lineárnym systémom rozumieme množinu*

$$\{Ax = b;\ x \in \mathbb{R}^n, A \in \boldsymbol{A},\ b \in \boldsymbol{b}\}.$$

Taktiež sa často využíva skrátené značenie

$$\mathbf{A}x = \mathbf{b}.$$

Definícia 12 (Množina riešení)**.** *Nech* $\boldsymbol{A}x = \boldsymbol{b}$ *je sústava intervalových lineárnych rovníc. Potom množinu*

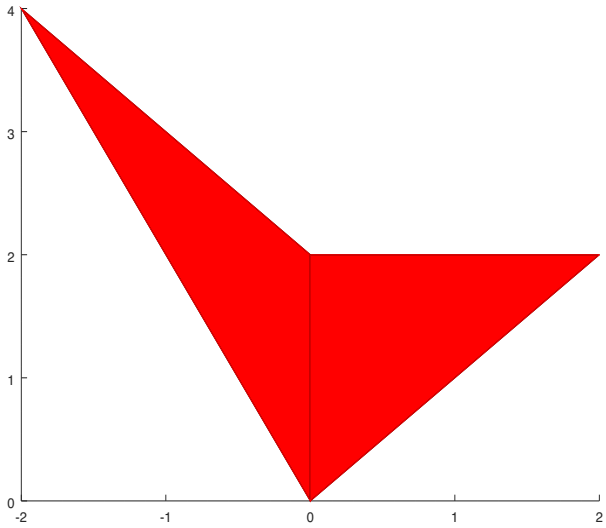
$$\Sigma = \{x \in \mathbb{R}^n;\ \exists A \in \boldsymbol{A},\ \exists b \in \boldsymbol{b} : Ax = b\},$$

nazývame množinou riešení sústavy intervalových lineárnych rovníc $\boldsymbol{A}x = \boldsymbol{b}$.

Množinou riešení sústavy intervalových lineárnych rovníc budeme rozumieť zjednotenie všetkých riešení zo všetkých reálnych systémov obsiahnutých v intervalovom systéme. Táto množina nemusí byť konvexná, je však konvexná v každom ortante [2]. Zistenie tejto množiny je vo všebecnosti NP-ťažký problém. Príkladom môže byť nasledujúci príklad z knihy Kulpa a kol. [11].

$$\begin{pmatrix} [0,1] & 1 \\ -2 & [-1,2] \end{pmatrix} x = \begin{pmatrix} [0,2] \\ 0 \end{pmatrix},$$

na obrázku 2.1 je znázornené grafické riešenie tohoto systému.



Obr. 2.1: Množina riešení intervalového systému.

Na obrázku 2.1 je dobre vidieť, že množina riešení intervalového systému môže nadobúdať rôzne zložité tvary. Preto pri intervalových systémoch je často našim cieľom nájsť aspoň nejaký n-rozmerný box, v ktorom daná množina riešení bude ležať. Inak povedané, budeme sa snažiť získať intervalový vektor, ktorý čo najtesnejšie ohraničí množinu riešení.

Definícia 13 (Intervalový obal)**.** *Nech Σ je obmedzená neprázdna množina riešení intervalového lineárneho systému $\boldsymbol{A}x = \boldsymbol{b}$, kde $\boldsymbol{A}$ je matica rozmerov $m \times n$. Potom intervalovému vektoru $\boldsymbol{h} = [\underline{h}, \overline{h}]$, definovanému*

$$\underline{h}_i = min_{x \in \Sigma} x_i \qquad (i = 1 \ldots n),$$

$$\overline{h}_i = max_{x \in \Sigma} x_i \qquad (i = 1 \ldots n),$$

kde x_i je i-ty koeficient x, hovoríme intervalový obal.

Definícia 14 (Vnútorná intervalová obálka)**.** *Nech Σ je množina riešení intervalového lineárneho systému $\boldsymbol{A}x = \boldsymbol{b}$. Intervalovému vektoru $[\underline{x}, \overline{x}]$ hovoríme vnútorná obálka množiny Σ, ak platí*

$$[\underline{x}, \overline{x}] \subseteq \Sigma.$$

Definícia 15 (Vonkajšia intervalová obálka)**.** *Nech Σ je množina riešení intervalového lineárneho systému $\boldsymbol{A}x = \boldsymbol{b}$. Intervalovému vektoru $[\underline{x}, \overline{x}]$ hovoríme vonkajšia obálka množiny Σ, ak platí*

$$\Sigma \subseteq [\underline{x}, \overline{x}].$$

Definícia 16. *Nech máme intervalový systém $\boldsymbol{A}x = \boldsymbol{b}$. Potom povieme, že*

1. *je silno riešiteľný, ak $\forall A \in \boldsymbol{A}, \forall b \in \boldsymbol{b}\ \exists x_0 :\ Ax_0 = b$,*
2. *je slabo riešiteľný, ak pre nejaké $A \in \boldsymbol{A}$, $b \in \boldsymbol{b}$ $\exists x_0 :\ Ax_0 = b$.*

Definícia 17. *Nech máme reálny vektor x. Potom x sa nazýva (slabé) riešenie intervalového lineárneho systému, ak platí*

$$Ax = b \text{ pre nejaké } A \in \boldsymbol{A},\ b \in \boldsymbol{b}.$$

Veta 1 (Oettli-Prager, 1964 [22])**.** *Majme intervalový lineárny systém $\boldsymbol{A}x = \boldsymbol{b}$, kde*

$$\boldsymbol{A} = [A^c - A^\triangle, A^c + A^\triangle] \in \mathbb{IR}^{m\times n},$$
$$\boldsymbol{b} = [b^c - b^\triangle, b^c + b^\triangle] \in \mathbb{IR}^{m}.$$

Potom x je slabé riešenie: $x \in \Sigma \Leftrightarrow |A^c x - b^c| \leq A^\triangle |x| + b^\triangle$

Dôkaz. „$\Rightarrow$" Nech $x \in \Sigma$, potom podľa definície 12. platí, že $\exists A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}$: $Ax = b$.

$|A^c x - b^c| = |(A^c - A)x + (Ax - b) + (b - b^c)| = |(A^c - A)x + (b - b^c)| \leq$
$\leq |(A^c - A)x| + |b - b^c| \leq |A^c - A||x| + |b - b^c| \leq A^\triangle |x| + b^\triangle$.

„$\Leftarrow$" Buď $|A^c x - b^c| \leq A^\triangle |x| + b^\triangle$. Definujeme vektor $y \in [-1,1]^n$

$$y_i = \begin{cases} \frac{(A^c x - b^c)_i}{(A^\triangle |x| + b^\triangle)_i}, & \text{pokiaľ } (A^\triangle |x| + b^\triangle)_i > 0, \\ 1, & \text{inak.} \end{cases}$$

Dostávame

$$(A^c x - b^c)_i = y_i (A^\triangle |x| + b^\triangle)_i,$$

čo nám následne dáva

$$A^c x - b^c = D(y)(A^\triangle |x| + b^\triangle),$$

kde $D(y)$ je diagonálna matica, ktorá má na diagonále y.
Ak označíme $z = \operatorname{sgn}(x)$, potom evidentne $|x| = D(z)x$. Odstránením absolútnej hodnoty z predchádzajúceho vzorca týmto spôsobom dostávame

$$A^c x - b^c = D(y)A^\triangle D(z)x + D(y)b^\triangle.$$

Prevedením prvkov obsahujúcich x na rovnaké strany dostávame

$$(A^c - D(y)A^\triangle D(z))x = b^c + D(y)b^\triangle.$$

Lahko nahliadneme, že

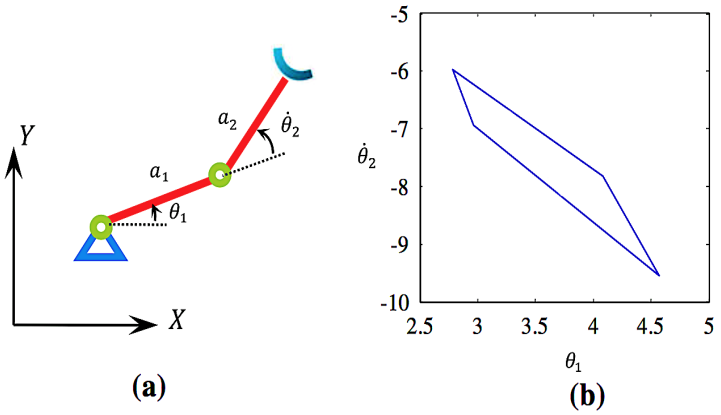
$$A^c - D(y)A^\triangle D(z) \in \boldsymbol{A},\ b^c + D(y)b^\triangle \in \boldsymbol{b},$$

čiže $x \in \Sigma$.

□

2.3 Parametrické intervalové lineárne sústavy

Parametrické intervalové lineárne sústavy sa vyskytujú v rôznych praktických aplikáciách ako sú napríklad robotové manipulátory. Vzťah medzi koncovou efektórovou pozíciou a posunom kĺbov je známy, nie je však nevyhnutne presný. Musíme počítať s určitými neistotami ako výrobné odchýlky mechanických častí, chyby merania, riadenia, či zaokrúhľovania. Vieme teda získať množinu, v ktorej sa nachádzajú všetky riešenia. Inými slovami, táto množina znázorňuje oblasť, do ktorej sa sériový manipulátor môže dostať posunom kĺbu.



Obr. 2.2: [14] **(a)** Dvojdimenzionálny sériový manipulátor. **(b)** Množina riešení dosahu ramena.

Definícia 18. *Nech* $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k$ *sú reálne parametre,* $A^k \in \mathbb{R}^{n\times n}$, $b^k \in \mathbb{R}^m$, $k = 0, \ldots, n$ *sú koeficienty na pozícii k-teho parametru. Píšeme*

$$A(\boldsymbol{p})x = b(\boldsymbol{p}),$$

kde

$$A(\boldsymbol{p}) = A^0 + \sum_{k=1}^{m} A^k \boldsymbol{p}_k, \quad b(\boldsymbol{p}) = b^0 + \sum_{k=1}^{m} b^k \boldsymbol{p}_k,$$

sa nazýva parametrický intervalový lineárny systém.

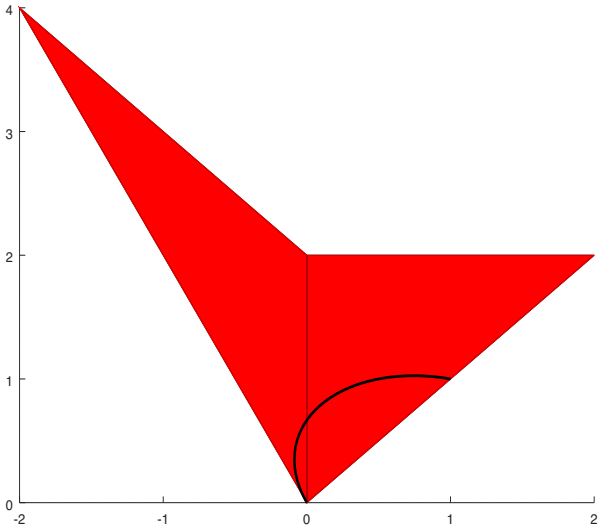
Množina riešení parametrickej intervalovej lineárnej sústavy je definovaná ako [18]

$$\Sigma^p = \{x \mid A(p)x = b(p), p \in \boldsymbol{p}\}.$$

Nasledujúci príklad je rovnaký ako v podkapitole 2.2 z knihy Kulpa a kol. [11]. S tým rozdielom, že intervaly v sústave sme nahradili intervalovým parametrom.

$$\begin{pmatrix} p & 1 \\ -2 & 3p-1 \end{pmatrix} x = \begin{pmatrix} 2p \\ 0 \end{pmatrix}, p \in [0,1]$$

Na obrázku 2.3 je znázornené grafické riešenie tohoto systému ako čierna krivka. Pre lepšiu predstavu si zobrazujeme aj množinu riešení intervalového systému z predchádzajúcej podkapitoly.



Obr. 2.3: Množina riešení parametrického intervalového systému.

3. Metóda branch and bound

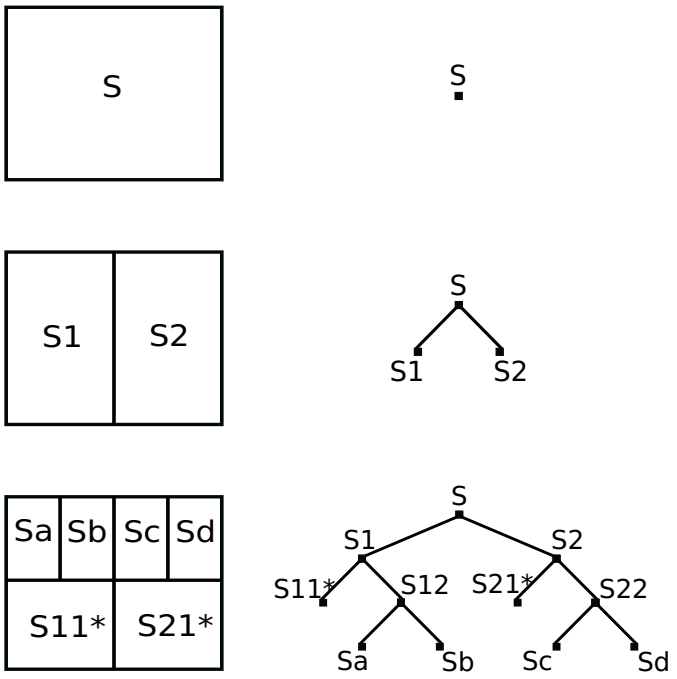
3.1 Základný popis

Nie celý počiatočný vyhľadávací priestor obsahuje riešenie. Tento priestor nájdeme tým, že riešime parametrický systém 2.3. Začíname počiatočným vyhľadávacím priestorom, pričom neprehľadávame na celom priestore. Vieme, že riešenie leží niekde v ňom. Buď máme tip, kde by potenciálne mohlo ležať alebo môžeme jeho polohu približne zistiť z podstaty problému. Napríklad, nemá zmysel riešenie sústavy hľadať medzi zápornými číslami, ak vopred vieme, že má byť kladné. Názorne si to môžeme predstaviť z obrázku 2.2. Počiatočný priestor tam máme vyhradený intervalmi [2,5; 5] a [−10; −5]. Vieme, že v tomto priestore sa nachádza riešenie, nie však v jeho celej časti. Našim cieľom je zredukovať tento priestor, na čo využijeme metódy branch and bound a branch and prune. Obe techniky slúžia na zníženie počtu možných riešení, ktoré je potrebné preskúmať v prípade, že existuje vyhľadávací priestor. Pozostávajú z troch základných častí:

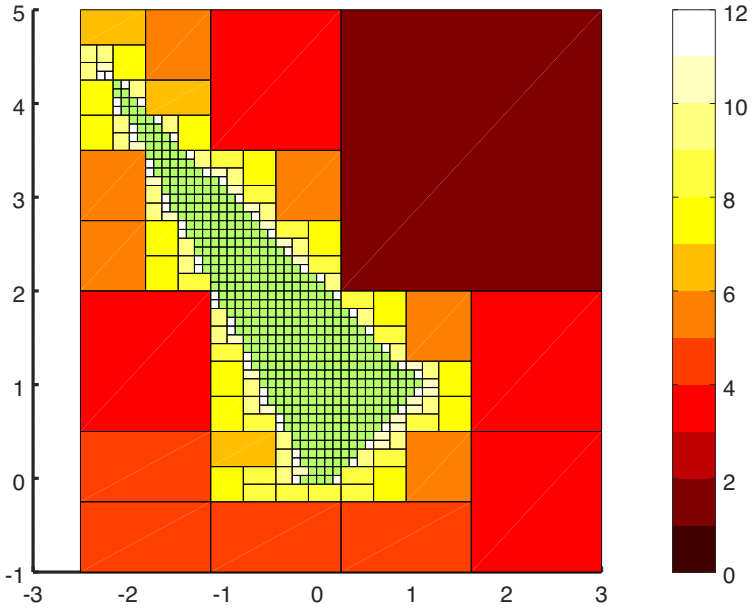
- **branching** - je rekurzívne delenie vyhľadávacieho priestoru na menšie podpriestory (disjunktné podmnožiny), ktoré sa preskúmajú v ďalších iteráciách. Delenie nastáva z toho dôvodu, že o počiatočnom boxe nevieme nič povedať a preto ho rozdelíme na menšie boxíky, o ktorých sa budeme snažiť získať nejaké informácie separátne. Následne na získané boxíky aplikujeme bounding alebo pruning metódu.

- **bounding** - môžeme chápať ako kontraktor. Snažíme sa obmedziť riešenie v danom boxíku. Informáciu neprenášame do riešenia v zmysle, že by sme hľadali globálne maximum, ale môžeme ju použiť pri hľadaní intervalového obalu. Chápeme ho vlastne ako globálne minimum a maximum vo všetkých smeroch, ktoré nám obmedzujú množinu riešenia.

- **pruning** - snaží sa zahodiť celý box. Snažíme sa o to pomocou rôznych prerezávacích podmienok, ktoré si bližšie definujeme v sekcii 4. Máme dopredu daný rozmer boxu, ktorý je závislý na danom probléme, čiže máme zadanú stopovaciu hladinu. Počiatočný box delíme na menšie a rozhodujeme, či obsahujú riešenie alebo nie. Ak nie, tak môžeme celú časť rovno zahodiť, čo je cieľom tejto časti. Ak riešenie obsahuje, tak delíme ďalej až kým nedosiahneme vopred stanovenú hranicu veľkosti boxu. Vtedy nastáva koniec delenia.



Obr. 3.1: Ilustrácia vyhľadávacieho priestoru, * − neobsahujú optimálne riešenie.



Obr. 3.2: Znázornenie hĺbky orezávania.

Predchádzajúci obrázok 3.2 farebne znázorňuje hĺbku ako aj postup delenia v konkrétnom príklade, ktorý sme uviedli v 2. kapitole. Znázorňuje nám koľkokrát musíme počiatočný box rozdeliť na jednotlivé boxy, až kým ich zahodíme. Zelené boxy nám určujú výslednú množinu. Zvyšná časť reprezentuje boxy, ktoré boli počas delenia zahodené, pričom sú zafarbené podľa hĺbky. Ľahko si môžeme všimnúť, že ako prvý bol oddelený najväčší box, znázornený tmavočervenou farbou.

3.2 Všeobecná procedúra

V nasledujúcej procedúre si môžeme všimnúť branch and bound algoritmus v podobe, v akej ho používame v našich metódach. Využívame rôzne iteračné metódy prerezávania pre intervalové systémy, s ktorými sa bližšie oboznámime v 4. kapitole.

Algoritmus 1 Všeobecná branch and bound procedúra

```
1: Solution = ∅
2: procedure BAB(x)
3:     bound_1
4:     ⋮                                          ▷ kontrahovacia sekcia
5:     bound_l
6:     prune_1
7:     ⋮                                          ▷ orezávacia sekcia
8:     prune_k
9:     if x ≠ ∅ then
10:        if size(x) > ϵ then
11:            [x_1, x_2] ← split(x)
12:            BAB(x_1)
13:            BAB(x_2)
14:        else
15:            Solution ← x                       ▷ pridanie do riešenia
16:        end if
17:    end if
18: end procedure
```

Počiatočný box delíme v polovici na dve časti. Delenie nastáva podľa najdlhšej strany a v prípade rovnosti strán vyberáme prvú v poradí, až kým nedostaneme koeficient veľkosti ϵ. Voľba veľkosti tohoto koeficientu záleží od toho ako presnú mriežku prerezávania požadujeme a zároveň podľa našich časových možností. Pre malé ϵ môže delenie trvať príliš dlho. Jeho veľkosť si prvotne volíme ako 1/100 alebo 1/50 šírky počiatočného vyhľadávacieho priestoru.

V prvotnej implementácii sme uvažovali aj iné pomery delenia, napríklad jedna štvrtina, tri štvrtiny alebo aj náhodný pomer delenia, dokonca v niektorých prípadoch bolo delenie v jednej štvrtine výhodnejšie. Avšak, nakoniec sa ukázalo, že všeobecne ideálny pomer neexistuje a najstabilnejšie výsledky dostaneme delením v jednej polovici. Z tohoto dôvodu budeme ďalej používať delenie na polovice.

4. Prerezávacie podmienky

V tejto kapitole uvedieme prerezávacie podmienky. Využívame ich v kombinácii s metódou branch and bound na získanie množiny riešení parametrických intervalových lineárnych sústav.

4.1 Priama relaxačná podmienka

Táto podmienka na základe svojej jednoduchosti funguje prekvapivo stabilne a dobre. Funguje na princípe, v ktorom sa zbavujeme lineárnej závislosti jednotlivých parametrov. Inými slovami, využívame intervalovú relaxáciu, ktorej základom je vyčíslenie výrazov pomocou intervalovej aritmetiky

$$\boldsymbol{A} = A^0 + \sum_{k=1}^{m} A^k \boldsymbol{p}_k, \quad \boldsymbol{b} = b^0 + \sum_{k=1}^{m} b^k \boldsymbol{p}_k.$$

Následne hľadáme riešenie výrazu $\boldsymbol{A}x \cap \boldsymbol{b} = 0$. Podľa Beeckovej vety vieme, že ak $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$, $\boldsymbol{b} \in \mathbb{IR}^m$, tak platí $\Sigma(\boldsymbol{A},\boldsymbol{b}) = \{x \in \mathbb{R}^n \mid \boldsymbol{A}x \cap \boldsymbol{b} \neq 0\} = \{x \in \mathbb{R}^n \mid 0 \in \boldsymbol{A}x - \boldsymbol{b}\}$. Túto vetu nájdeme aj s dôkazom v [15]. V našej implementácii túto podmienku uvádzame pod názvom `prune_intersect`.

4.2 Využitie hodnosti

Základom je veta, ktorá hovorí, že ak máme dané $A \in \mathbb{R}^{m\times n}$ a existuje matica $R \in \mathbb{R}^{n\times m}$ taká, že $\|I - RA\|_1 \leq 1$, kde I je jednotková matica a $\|\cdot\|$ je norma, potom A má lineárne nezávislé stĺpce [23]. Inými slovami, matica A má hodnosť n. Aby sme ju mohli využiť, musíme ju zovšeobecniť pre normu intervalvých matíc. Maximovú normu si definujeme ako

$$\|\boldsymbol{x}\|_1 := \max\{|\boldsymbol{x}_i| \mid i = 1, \ \ldots, \ n\},$$

$$\|\boldsymbol{A}\|_1 := \max\{\sum_{k=1}^{n} |\boldsymbol{A}_{ik}| \mid i = 1, \ \ldots, \ m\},$$

kde $\boldsymbol{x} \in \mathbb{IR}^n$ a $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ [15].
Teda z tohoto nám vyplýva, že

$$\|I - R\boldsymbol{A}\|_1 \leq 1.$$

Dôsledkom toho sústava $\boldsymbol{A}x = \boldsymbol{b}$, kde $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$, $\boldsymbol{b} \in \mathbb{IR}^m$ je neriešiteľná, pokiaľ má intervalová matica $[\boldsymbol{A}|\boldsymbol{b}]$ hodnosť rovnú n+1. V prípade, že podmienka neplatí, nie sme schopní o neriešiteľnosti nič povedať. V implementácii ju uvádzame ako `prune_fcr`.

4.3 Reziduálny prístup

Vyjdeme zo symbolov popísaných v [10]

$$x^* := A(p^c)^{-1} b(p^c),$$

$$x := x^* + y.$$

Následne parametrický intervalový lineárny systém prepíšeme takto:

$$A(\boldsymbol{p})y = b(\boldsymbol{p}) - A(\boldsymbol{p})x^*,$$

kde ako vidíme, neznámou je y. Z výrazu $x^* + \boldsymbol{y}$ dostaneme výsledný obal $\boldsymbol{x}$ pôvodného riešenia. Využijeme, že v intervalovej aritmetike sčítanie a násobenie nie je distributívne, ale je subdistributívne a preto dostávame výraz

$$(\sum_{k=1}^{m} \boldsymbol{p}_k A^k)y = \sum_{k=1}^{m} \boldsymbol{p}_k(b^k - A^k x^*).$$

Následne ho zrelaxujeme na intervalovú lineárnu sústavu bez parametrov, potom možno bude mať užšie intervaly u parametrov ako keby sme relaxovali priamo. V našej funkcii s názvom `prune_intersect2` hľadáme riešenie výrazu $\boldsymbol{A}y \cap \boldsymbol{b} = 0$.

4.4 Oettli-Prager

E. Popova publikovala zreteľný popis množiny riešení sústav intervalových lineárnych rovníc (definícia 12) za špeciálnych podmienok v roku 2009 [19]. Veľmi sa podobá na vetu 1. Oettli-Prager zo sekcie 2.2. Implementovali sme ju v podobe, v akej ju popísal Hladík [7]. Konkrétne sa jedná o vetu:

Veta 2. *Majme $x \in \mathbb{R}^n$ a $\boldsymbol{p} \in \mathbb{IR}^k$. Pokiaľ $x \in \Sigma^p$, potom je riešením nerovnice*

$$|A(p^c)x - b(p^c)| \leq \sum_{k=1}^{m} p_k^{\triangle} |A^k x - b^k|.$$

Túto vetu využívame v podobe, že pokiaľ platí obrátená implikácia, tak x neobsahuje riešenie. Je teda postačujúce, aby to platilo pre aspoň jednu nerovnicu z danej sústavy nerovníc. V implementácii `prune_oettli` naozaj využívame absolútne hodnoty ako sú uvedené vo vete 2., nielen magnitúdy ako by sa na prvý pohľad mohlo zdať.

4.5 Gauss-Seidel

Gauss-Seidelova metóda je modifikácia Jacobiho iterácie, kde pri výpočte i-tej súradnice v $(k+1)$-om aproximovanom vektore berieme do úvahy, že súradnice $1, 2, \ldots, (i-1)$ sme už vypočítali a preto ich využijeme namiesto $\boldsymbol{x}^{(k)}$ príslušných súradníc. Takže na vypočítanie komponentu $\boldsymbol{x}_i^{(k+1)}$ využijeme aj v tomto iterovanom kroku získané hodnoty. Iba v prípadoch, keď sú komponenty väčšie ako i-ta, používame hodnoty väčších komponentov ako $\boldsymbol{x}_i^{(k)}$. Pre prvý komponent v $\boldsymbol{x}^{(k+1)}$ využijeme Jacobiho iteráciu. Toto je štandardný postup na zlepšenie algoritmov, ale nie je isté, že v každom prípade to pomôže. Vzorec:

$$\boldsymbol{x_i^{(k+1)}} = \frac{\boldsymbol{b_i}}{\boldsymbol{a_{ii}}} - \sum_{l=1}^{i-1} \frac{\boldsymbol{a_{il}}}{\boldsymbol{a_{ii}}} \boldsymbol{x_l^{(k+1)}} - \sum_{l=i+1}^{n} \frac{\boldsymbol{a_{il}}}{\boldsymbol{a_{ii}}} \boldsymbol{x_l^{(k)}},$$

kde $i = 1, 2, \ldots, n$; $k = 1, 2, 3, \ldots -$ je index iterácie a $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$ sú začiatočné hodnoty.

Tento postup si môžeme predstaviť ako kontraktor, ale môže nastať aj situácia, že ak dva po sebe idúce vektory majú prázdny prienik, tak box zahadzujeme. Výhoda tejto metódy je, že novospočítané koeficienty vektoru obálky počítame rovno a nie až v ďalšej iterácii ako u Jacobiho metódy. Medzi nevýhodu radíme, že boxy nie sú zarovnané, ale sú rôzne usporiadané. Nemusia mať rovnaké osi. Táto metóda pracuje ako tzv. bounding zo sekcie 3.

4.6 Hladík

Nasledujúca veta spolu s pomocnou Farkasovou lemmou nám umožňuje implementáciu metódy `prune_hladiklp`. Túto vetu sme získali vďaka poznámkam doc. Hladíka, ktorá tvorí nenahraditeľnú súčasť tejto metódy.

Lemma 3 (Farkasova [22])**.** *Sústava $Ax \leq b$ je riešiteľná práve vtedy, keď pre každé $p \geq 0$ také, že $A^T p = 0$ platí, že $b^T p = 0$.*

Dôkaz. Ak x rieši sústavu $Ax = b$ a $A^T p = 0$ platí pre nejaké $p \geq 0$, potom $b^T p = p^T b \geq p^T Ax = 0$. Naopak, nech za rovnakých podmienok platí, že $A^T p \geq 0$ pre každé $p \geq 0$ a $A^T p \leq 0$ pre každé $b^T p \geq 0$. Avšak, podľa Farkasovej lemmy to znamená, že sústava

$$Ax_1 - Ax_2 + x_3 = b$$

je realizovateľná. Dôsledkom nezápornosti x_3 máme $A(x_1 - x_2) \leq b$ a teda sústava $Ax \leq b$ je riešiteľná.

□

Veta 4.

$$M(x) := (A^1 x - b^1 \mid A^2 x - b^2 \mid \cdots \mid A^K x - b^K).$$

Nech u^*, r^*, s^* sú optimálne riešenia lineárneho programu

$$\min\ e^T r + e^T s;\ M(x_c)^T u + r - s = 0,\ r, s \geq 0,\ \overline{p}^T r - \underline{p}^T s \leq -1.$$

Teraz, ak

$$1 + \textstyle\sum_k p_\Delta^k |(A^k \boldsymbol{x} - b^k)^T u^*| \leq \sum_k p_c^k (A^k \boldsymbol{x} - b^k)^T u^*,$$

potom $\boldsymbol{x} \cap \Sigma^p = \emptyset$.

Dôkaz. Sústava $A(p)x = b(p)$ má tvar $M(x)p = 0$. Podľa Farkasovej lemmy sústava

$$M(x)p = 0,\ \underline{p} \leq p \leq \overline{p}$$

je neriešiteľná v prípade, ak sústava

$$M(x)^T u + r - s = 0,\ r, s \geq 0,\ \overline{p}^T r - \underline{p}^T s < 0$$

je riešiteľná. Prepíšeme to ako

$$M(x)^T u + r - s = 0,\ r, s \geq 0,\ p_c^T (r - s) + p_\Delta^T (r + s) < 0.$$

Nahradením $v := s - r$ vieme, že $|v| \leq r + s$, potom

$$M(x)^T u = v, \ -p_c^T v + p_\triangle^T |v| < 0.$$

Nahradením za v máme

$$p_\triangle^T |M(x)^T u| < p_c^T M(x)^T u.$$

Nahradením za $M(x)$ máme

$$\sum_k p_\triangle^k |(A^k x - b^k)^T u| < \sum_k p_c^k (A^k x - b^k)^T u.$$

Intervalovaním tejto nerovnosti sme hotoví.

□

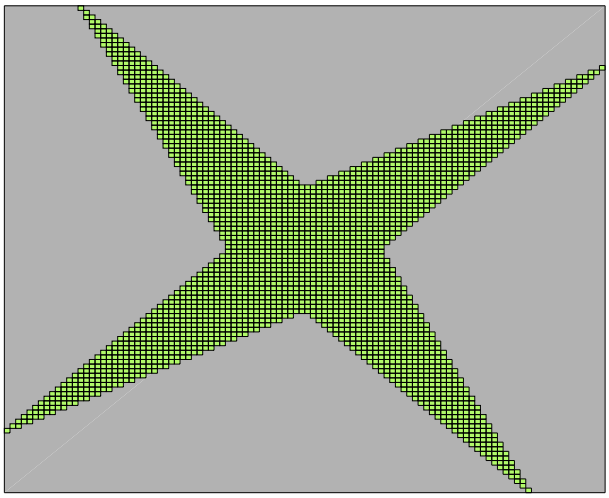
Poznamenajme, že optimálne riešenie lineárneho programu nemusí byť overené. V skutočnosti môžeme vziať nejaké iné u^*, ale navrhované vyzerá sľubne.

5. Spracovanie získaných boxov

V nasledujúcej kapitole si predstavíme niekoľko spôsobov spracovania obdržaných riešení. Pokúsime sa vysvetliť ako dané algoritmy pracujú a bližšie ukázať v čom spočívajú ich hlavné výhody a naopak aj ich nevýhody. Budeme sa zaoberať spôsobmi spracovania výsledných intervalových boxov z použitej branch and bound metódy. Tieto boxy v sebe obsahujú dôležitú informáciu a to riešenie pre danú sústavu. Ukážeme si hneď tri vonkajšie spôsoby ako pracovať so spomínanými boxmi: jeden, ktorý využíva kvadranty a dva nasledujúce opisujúce získané riešenia. Jeden z nich navyše využíva jednoduchú heuristiku. Cieľom všetkých spomínaných spôsobov je minimalizácia počtu boxov a prípadne ich vhodné zlievanie, pre čo najpresnejší výsledok. Poznamenajme, že algoritmy vieme zovšeobecniť aj pre väčšie dimenzie, ale my pre názornosť budeme pokračovať len v dvoch dimenziách. Využitie spomínaných algoritmov spočíva najmä v zjednodušení množiny riešení, taktiež nám urýchľujú vykresľovanie intervalových riešení a sú vhodné na použitie v ďalších možných experimentoch.

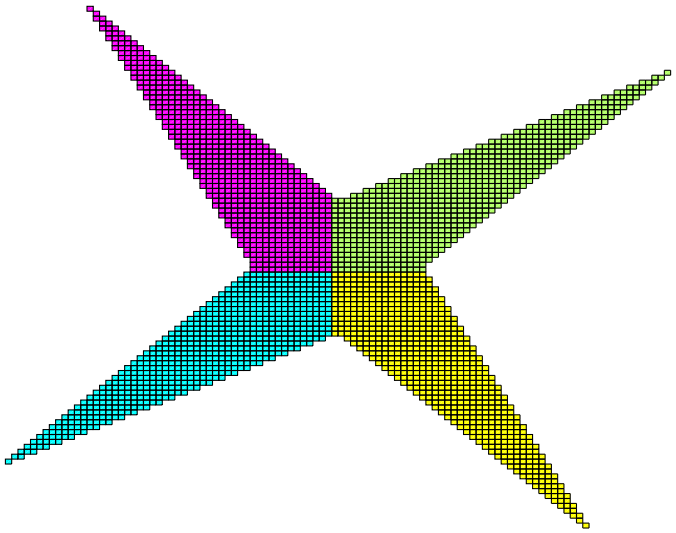
5.1 Štvrtinová metóda

Budeme sa snažiť množinu riešení čo najpresnejšie ohraničiť z vonkajšej strany. Preto tento algoritmus začne práve tým, že množinu riešení ohraničí jedným veľkým boxom (sivý obdĺžnik na obrázku 5.1).



Obr. 5.1: Vonkajšie ohraničenie danej množiny.

Následne si tento vzniknutý veľký box prerozdelíme do kvadrantov, takže dostaneme štyri menšie boxy (kvadranty). Potom si pomocou týchto boxov rozdelíme množinu riešení do štyroch skupín podľa toho, do ktorého kvadrantu patria, čo zistíme tak, že s ním tvoria prienik. Boxy, ktoré ležia na rozhraní viacerých kvadrantov prerozdelíme tak, že do každého príslušného kvadrantu dáme len tú časť boxu, s ktorým tvorí prienik. Už máme teda rozdelené boxy podľa kvadrantov, čo je názorne ukázané na nasledujúcom obrázku 5.2, v ktorom sú farebne prerozdelené jednotlivé skupiny.



Obr. 5.2: Rozdelenie boxov podľa kvadrantov.

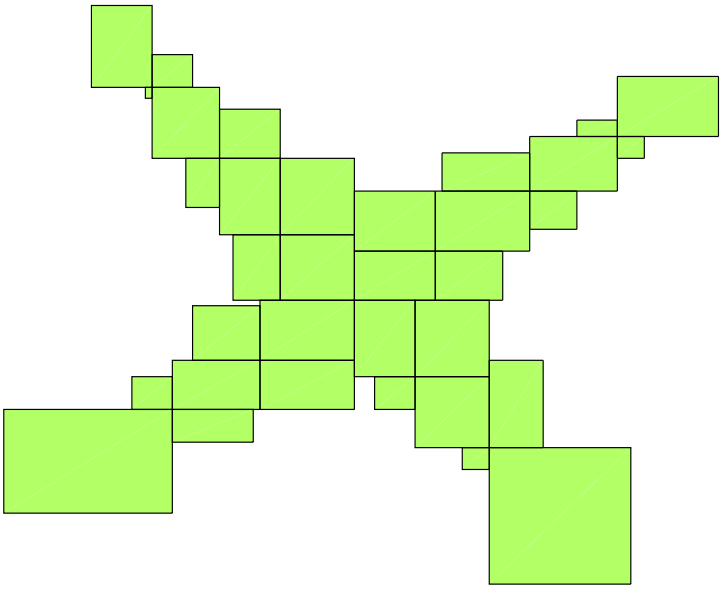
Teraz si znovu obalíme každú jednu vzniknutú skupinu riešení jedným boxom, podobne ako na začiatku vstupnú množinu riešení. Teda už máme každý kvadrant množiny riešenia obalený boxom.

Tento proces - prerozdeľovanie kvadrantov na ďalšie kvadranty, ich obaľovanie boxmi rekurzívne opakujeme, až kým veľkosť strany niektorého z kvadrantov nebude menšia ako vopred stanovený limit na šírku, ktorý si volíme sami.

Hlavnou výhodou tohoto algoritmu je to, že si môžeme vopred určiť počet chcených výsledných boxov vďaka stanoveniu limitu.

Algoritmus 2 Štvrtinová metóda

1: Solution = $\emptyset$
2: **procedure** OUTERBOUND(boxes, ϵ)
3: $[S_1, S_2, S_3, S_4] \leftarrow$ quadsol(boxes) ▷ rozdelenie boxov do 4 skupín
4: $\forall S_i : i = 1, \ldots, 4$
5: $U_i \leftarrow \underline{\bigcup} S_i$ ▷ každú skupinu obalím intervalovým obalom
6: **if** max $w(U_i) > \epsilon$ **then**
7: Solution $\leftarrow$ OUTERBOUND(S_i, ϵ)
8: **else**
9: Solution $\leftarrow U_i$ ▷ pridanie do riešenia
10: **end if**
11: **end procedure**



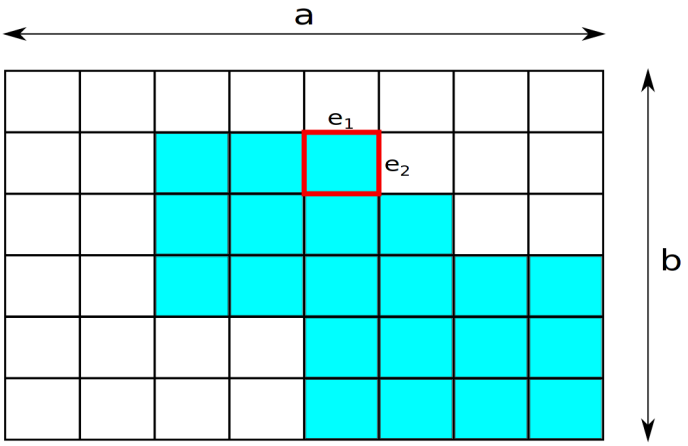
Obr. 5.3: Názorný príklad fungovania algoritmu `outerbound` po troch rekuzívnych deleniach.

Za najväčšiu výhodu považujem, že tento algoritmus bez akýchkoľvek zmien vieme použiť na výsledné množiny, ktoré dostaneme z parametrických intervalových lineárnych sústav. Riešenie týchto sústav často obsahuje až príliš veľa boxov, ktoré sa dokonca aj zbytočne prekrývajú. Nastavením správneho limitu pre šírku kvadrantu tak vieme dostať naozaj zaujímavé výsledky za pomerne krátky čas a hlavne výslednú množinu v menšom počte boxov.

5.2 Zlievanie boxíkov

Základná myšlienka tohoto algoritmu je spätné odkrokovanie štýlu s akým sme pôvodne delili, tj. delenie podľa najdlhšej strany. V prípade rovnakej dĺžky strán delenie nastáva podľa prvej strany, čím sa zabezpečí determinizmus. Delenie boxov teda nemôže byť náhodné, je nutné poznať jeho presný postup. Algoritmus vieme prepísať aj pre iné deterministické spätné delenie boxíkov.

Rozmery počiatočnej oblasti prerezávania a veľkosti jednotlivých boxíkov, ktoré budeme zlievať už poznáme. Pomocou týchto informácií si budeme reprezentovať boxíky v bitovom poli, tzv. bitmape. Tieto boxíky nám vznikli delením na polovice, čiže ich máme párny počet v každom smere. Následne si zistíme rozmery bitmapy. Vyberieme si najmenší boxík, ktorý máme a zistíme koľko krát sa jeho hrana zmestí do celkovej počiatočnej oblasti prerezávania. Túto hranu berieme po zverifikovaní, tj. po zväčšení tak, aby sme nestratili žiadnu množinu riešení. Veľkosť počiatočnej oblasti prerezávania si označme $a \times b$, veľkosť najmenšieho boxíku $e_1 \times e_2$. Rozmery bitmapy teda budú $\frac{a}{e_1} \times \frac{b}{e_2}$.



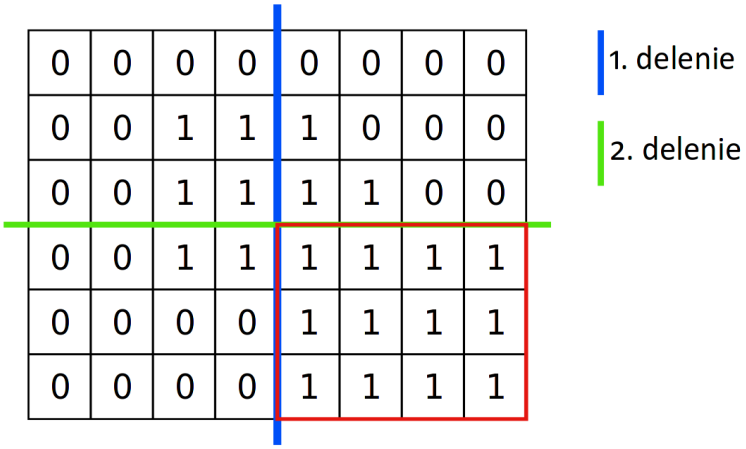
Obr. 5.4: Ilustrácia množiny riešení v počiatočnej oblasti prerezávania.

Týmto už máme vytvorenú bitmapu, čiže šablónu, s ktorou budeme ďalej pracovať. Vyplníme si ju nulami a jednotkami podľa toho, či sa v danej časti nachádza niektorý z našich pôvodných boxíkov alebo len časť z nich. Jednotku dávame, ak sa tam nachádza, nulu, ak nie. Takže už máme vyplnenú celú bitmapu.

0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

Obr. 5.5: Ilustrácia vyplnenej bitmapy.

V ďalšom kroku budeme chcieť boxíky spätne zlievať. Využijeme na to metódu branch and bound zhora. Rozdelíme si bitmapu podľa najdlhšej hrany a následne sa pomocou algoritmu pozrieme na oblasti, ktoré vznikli po delení a zistíme, či je celá oblasť vyplnená jednotkami. Ak áno, tak môžeme hneď povedať, že táto oblasť je celý jeden boxík. Ďalej ju už nedelíme a jej rozmery (súradnice) si zaznamenáme do zásobníku. V opačnom prípade, ak nie je vyplnená samými jednotkami, tak rekurzívne delíme ďalej, až kým nedostaneme oblasť iba s jednotkami, prípadne, až kým sa nedostaneme na úroveň veľkosti najmenšieho boxíku. Môžeme poznamenať, že takýto prípad nastáva väčšinou na okrajoch množiny. Počas celého procesu si pamätáme najdlhšiu hranu, podľa ktorej delíme.



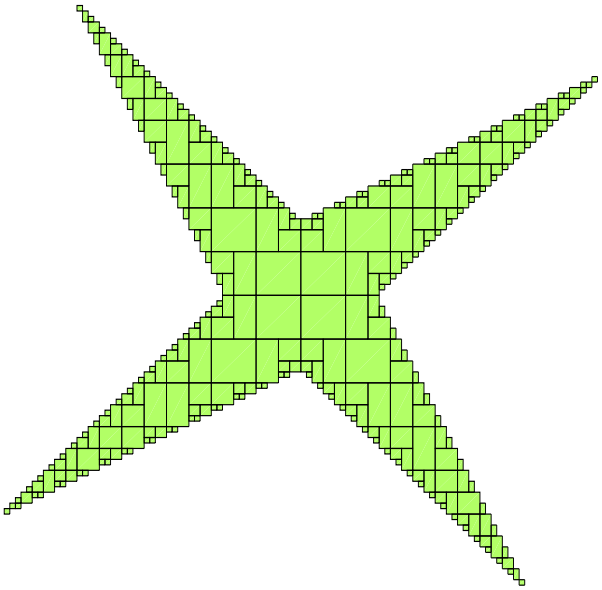
Obr. 5.6: Ilustrácia rozdelenia vyplnenej bitmapy.

Dostali sme sa teda do bodu, kedy už máme celý zásobník so súradnicami, ktoré reprezentujú pozlievané boxíky. Pomocou reverzného procesu sa dá z bitmapy jednoducho prejsť späť k boxom, ale musíme si dať pozor na zaokrúhľovanie. Získame ich vlastne vďaka údajom o najmenšom boxíku, z ktorého sme získali polia a rozmery v bitmape a vďaka súradniciam boxov v zásobníku.

Za výhodu tohoto algoritmu považujeme, že výsledné pozlievané boxy sú zarovnané v jednej úrovni, kvôli spätnej konštrukcii branch and bound algoritmu.

Algoritmus 3 Zlievanie boxíkov

```
1: newboxes = ∅
2: solution = ∅
3: procedure MERGEBOXES(boxes, initx)
4:     map ← creatmap(boxes, initx)              ▷ vytvorenie bitmapy
5:     Stack ← map                               ▷ naplnenie zásobníku
6:     while Stack ≠ ∅ do
7:         s ← Stack
8:         if allones(s) then                    ▷ overenie výberu, samé jednotky
9:             solution ← s
10:        else
11:            Stack ← split(s)                  ▷ delenie na ďalšie oblasti
12:        end if
13:    end while
14:    for all  k ∈ solution do
15:        newboxes ← returningMerged(k)         ▷ reverzný proces späť k boxom
16:    end for
17: end procedure
```

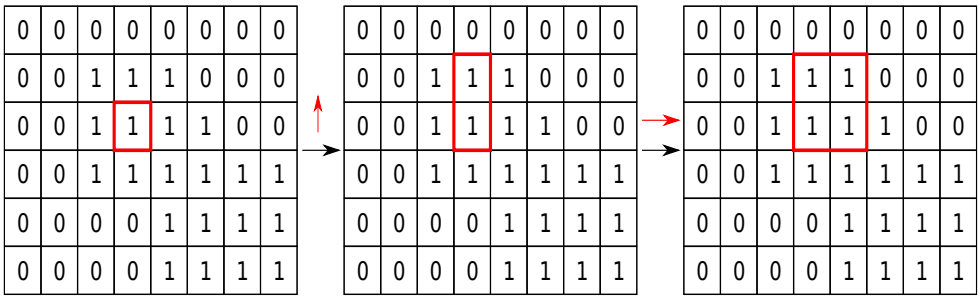


Obr. 5.7: Názorný príklad fungovania algoritmu `mergeboxes`.

5.3 Zlievanie boxíkov s heuristikou

V tomto algoritme je potrebné si zostaviť bitmapu, rovnakú ako v predchádzajúcom prípade. Rozdiel nastáva až pri zlievaní boxíkov. Na to využíva nasledujúcu heuristiku.

Náhodne si vyberieme jeden boxík v bitmape, ale tak, aby v sebe obsahoval jednotku. Volíme náhodný výber, aby sme sa s veľkou pravdepodobnosťou „trafili" do stredu bitmapy. Výber z kraja je menej výhodný, pretože je tam veľa výsledných boxíkov, ktoré majú rozmery minimálneho boxíku. Následne si opäť náhodne vyberieme jeden z povolených smerov. Je to taký smer, v ktorom je políčko vyplnené jednotkou. Vydáme sa týmto smerom a tým zväčšíme doterajšie celé pole, teda rozširujeme štruktúru.



Obr. 5.8: Vývoj zlievania boxíkov.

Vyššie opísané vyberanie smeru opakujeme, pokým máme povolený smer pre celú štruktúru. Teda expandujeme, pokým v nejakom smere máme políčko vyplnené jednotkou. Keď sa dostaneme do bodu, že už nemáme možnosti na ďalšie rozširovanie, tak si súradnice danej štruktúry uložíme do zásobníku. Následne si vyberieme nový začiatok tohoto procesu zo zvyšných jednotiek v bitmape a re-

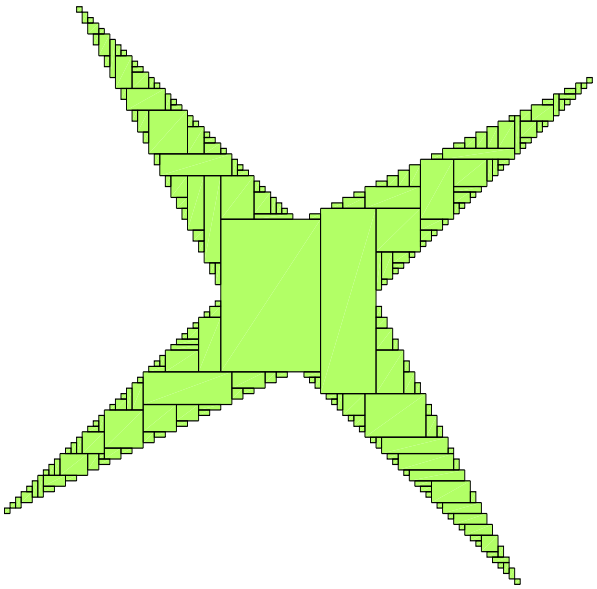
kurzívne sa pokúšame znovu rozšíriť štruktúru, až kým nedôjdeme ku krajným boxíkom (1x1).

Po vyčerpaní možností a získaní celého zásobníku si z neho prepočítame súradnice na výsledné boxy, podobne ako v predchádzajúcom algoritme.

Výhoda tohoto algoritmu spočíva v tom, že máme veľké plochy vo vnútri boxu a okraje, ktoré aj tak väčšinou nie sú presné, nás vôbec nemusia zaujímať. Ako nevýhodu vidíme to, že tento proces môže byť pomalý, najmä ak máme veľa políčok s jednotkami.

Algoritmus 4 Zlievanie boxíkov s heuristikou

```
1: newboxes = ∅
2: solution = ∅
3: procedure MERGEBOXESHEU(boxes, initx)
4:     map ← creatmap(boxes, initx)                    ▷ vytvorenie bitmapy
5:     while ∃ cell ∈ map & cell ∉ Solution do
6:         cell ← randChoiceOne(map)
7:         while spreading(cell) is true do            ▷ pokiaľ je zlučovanie možné
8:             cell ← randSpreading(cell)              ▷ náhodný smer pre zlučovanie
9:         end while
10:        solution ← cell
11:    end while
12:    for all k ∈ solution do
13:        newboxes ← returningMerged(k)               ▷ reverzný proces späť k boxom
14:    end for
15: end procedure
```



Obr. 5.9: Názorný príklad fungovania algoritmu `mergeboxesheu`.

6. Metóda branch and bound cez parametre

V tejto kapitole sa budeme zaoberať situáciou, keď sa nám vyskytne, že budeme mať menej parametrov ako je dimenzia premenných. Potom sa nám oplatí využiť branch and bound metódu cez parametre, pretože ich je menej. Čím viac dimenzií na branchovanie máme, tým proces trvá dlhšie, exponenciálne dlhšie. Keby bola dimenzia premenných menšia ako počet parametrov, tak by bolo lepšie túto metódu využívať cez dimenzie.

Algoritmus 5 Branch and bound procedúra cez parametre

```
1: Solution = ∅
2: procedure BAB(p)
3:     bound_1
4:     ⋮                                          ▷ kontrahovacia sekcia
5:     bound_l
6:     prune_1
7:     ⋮                                          ▷ orezávacia sekcia
8:     prune_k
9:     if p ≠ ∅ then
10:        if size(p) > ϵ then
11:            [p_1, p_2] ← split(p)
12:            BAB(p_1)
13:            BAB(p_2)
14:        else
15:            Solution ← p                       ▷ pridanie do riešenia
16:        end if
17:    end if
18: end procedure
```

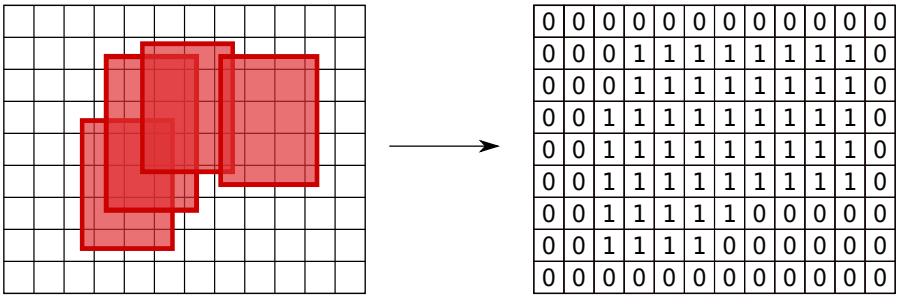
Počas algoritmu sa dostávame do stavu, kedy máme veľa malých boxov z parametrov, ktoré sú menšie ako naše ϵ. Nevieme však rozhodnúť, či obsahujú riešenie alebo nie. Je to podobné ako pri branch and bound metóde cez riešenie, len v tomto prípade sú tie boxy z parametrov. Z nich potrebujeme zistiť, kde leží riešenie. Preto je veľmi dôležité vziať všetky parametre a intervalizovať ich. Tento krok nájdeme len v metóde branch and bound cez parametre, nie cez riešenie. Z každého boxu teda spravíme intervalovú sústavu $\boldsymbol{A}x = \boldsymbol{b}$, ktorú vyriešime pomocou $\boldsymbol{A} \setminus \boldsymbol{b}$. Z toho sme schopní získať x, ktoré zapúzdruje množinu riešení.

Algoritmus 6 Získanie riešení z parametrov

```
1: Solution = ∅
2: [iA, ib] ← formmatrix(pk, Ak, bk, A, b)      ▷ získanie intervalvej sústavy
3: procedure ENCAPSULATED(iA, ib)
4:     enc ← iA\ib                  ▷ získam x, ktorý zapúzdruje riešenie
5:     Solution ← enc
6: end procedure
```

Ďalšia otázka, ktorá sa nám vynára je spôsob prerezávania. Na prvý pohľad sa nám ponúka podmienka využívajúca hodnosť matice, ktorá je vysvetlená v podkapitole 4.2, pretože nepracuje s celým počiatočným priestorom. Vlastne hovorí, že sústava $\boldsymbol{A}x = \boldsymbol{b}$ je neriešiteľná, pokiaľ má intervalová matica $[\boldsymbol{A}|\boldsymbol{b}]$ hodnosť rovnú $n + 1$. Podmienka, ktorá by taktiež mohla fungovať je Oettli-Prager 4.4.

Na konci algoritmu sa dostávame do situácie, kedy máme priveľa boxov, ktoré sa nám navyše veľmi prekrývajú. Môžeme na ne ale aplikovať niektoré algoritmy z kapitoly 5. Štvrtinovú metódu (5.1) môžeme využiť bez akýchkoľvek ďalších úprav. V prípade zlievacích metód môžeme využiť tú s heuristikou. Je však nutné pred použitím boxy nafúknuť do mriežky, keďže nemusia byť vopred do nej zaradené (obrázok 6.1). V podstate dáme jednotky aj tam, kde daný box len zasahuje. Následne už môžeme využiť zlievanie s heuristikou ako ho máme uvedené v podkapitole 5.3.



Obr. 6.1: Ilustrácia zaradenia boxov do bitmapy.

7. Testovanie

V tejto kapitole si uvedieme porovnanie jednotlivých prerezávacích podmienok na parametrických intervalových lineárnych sústavách s rôznym počtom parametrov. Uvedieme si jednotlivé sústavy, pri ktorých uvádzame aj zdroj a následne tabuľky, v ktorých sledujeme počet vrátených boxov jednotlivých prerezávacích podmienok za určitý čas. Čím ich je menej, tým sme bližšie k množine riešenia. Všetky prerezávacie podmienky porovnávame s rovnakým parametrom. Ten určuje šírku intervalu, ktorý je povolený na rozdelenie. Je dôležité si uvedomiť, že počet vrátených boxov môžeme porovnávať len preto, že nám vďaka rovnakému parametru ϵ vracajú približne rovnako veľké boxy. Jedine podmienka Gauss-Seidel nedelí dané boxy, len ich zmenšuje, prípadne zahodí a preto sa jej veľkosti boxov môžu odlišovať v porovnaní s ostatnými metódami. Následne si niektoré množiny riešení zobrazíme. Testovanie prebiehalo na prenosnom počítači s procesorom Intel Core i7 (3 GHz), 8GB RAM, Octave 4.2.1.

7.1 Jednoparametrická sústava

$$A(p) = \begin{pmatrix} p & 1 \\ -2 & 3p-1 \end{pmatrix}, \quad b(p) = \begin{pmatrix} 2p \\ 0 \end{pmatrix}, \quad p \in [0,1] \text{ [11]}.$$

Hranice počiatočného boxu: $([-2{,}5;\ 3], [-1;\ 5])^{\mathrm{T}}$.

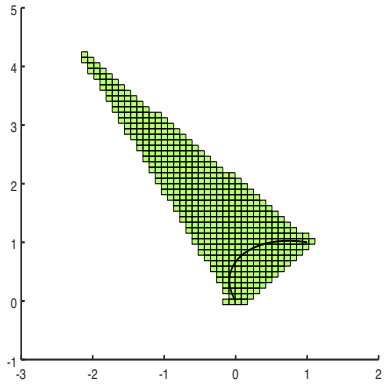
epsilon	**0,3**	**0,1**	**0,05**	**0,02**
	#/s	#/s	#/min	#/min
`intersect`	141/8	466/25	1669/1,4	24537/19
`intersect2`	233/11	824/38	3049/2,3	45882/34
`oettliPrager`	109/6	380/18	1427/0,9	21801/13
`gaussSeidel`	400/10	2031/48	7910/3,1	82752/32
`hladiklp`	71/9	132/28	271/1,5	1045/21

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

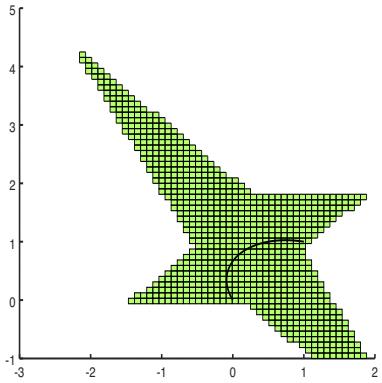
Tabuľka 7.1: Využitie branch and bound metódy v priestore riešení pre jednoparametrickú sústavu (Skalna1).

V nasledujúcich obrázkoch sú znázornené jednotlivé výsledné množiny pre každú prerezávaciu podmienku s parametrom šírky $\epsilon = 0{,}1$. Vidíme v nich, že metódy, v ktorých nahradíme parameter s daným intervalom ako napríklad `intersect` podmienka, výslednú množinu celkom nafúknu. Aj napriek tomu, výsledné množiny konvergujú k riešeniu intervalizovanej sústavy. Riešenie vykresľujeme do obrázkov čiernou krivkou. Ako už vyplýva z tabuľky 7.1, najbližšie sa k riešeniu dostala prerezávacia podmienka `hladiklp`, obrázok 7.7. Na obrázku 7.8 sme ju zobrazili aj pomocou funkcie `outerbound` s obmedzením 0,5. Môžeme si všimnúť, že sme počet boxov znížili približne na 1/5 vzhľadom k pôvodnej výslednej množine. Z fungovania funkcie `outerbound` vyplýva, že sme ju ale o niečo

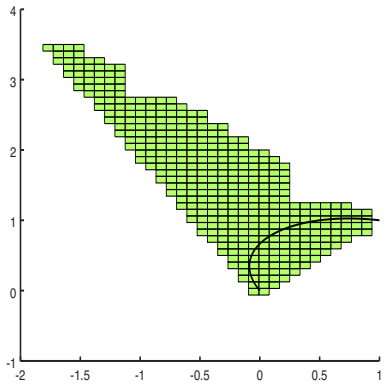
zväčšili. Záleží na konkrétnej úlohe a na samotnom užívateľovi, ktorý prístup k množine je výhodnejší.



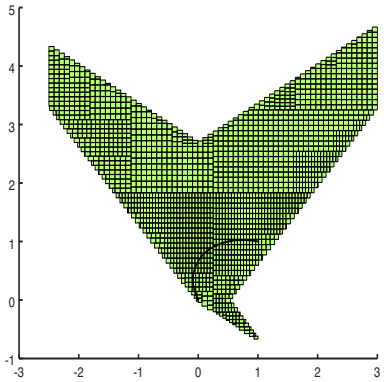
Obr. 7.1: `intersect` $\epsilon = 0{,}1$.



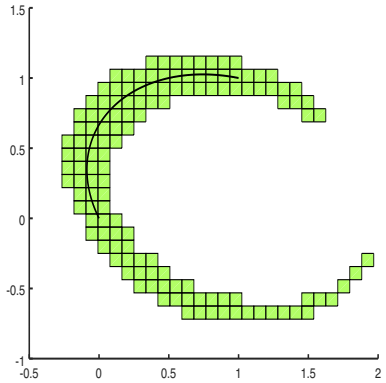
Obr. 7.2: `intersect2` $\epsilon = 0{,}1$.



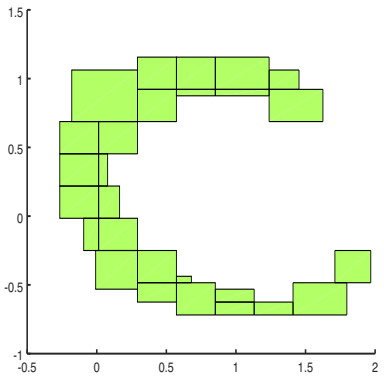
Obr. 7.3: `oettliPrager` $\epsilon = 0{,}1$.



Obr. 7.4: `gaussSeidel` $\epsilon = 0{,}1$.



Obr. 7.5: `hladiklp` $\epsilon = 0{,}1$.

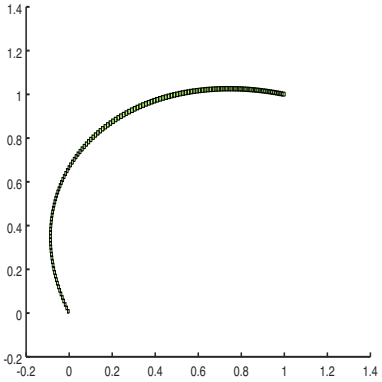


Obr. 7.6: `outerbound` $\epsilon_{ob} = 0{,}5$.

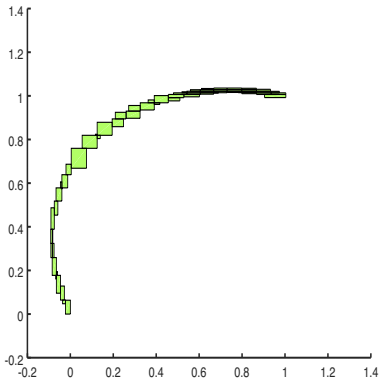
epsilon	**0,01**	**0,005**	**0,001**	**0,0005**
	#/s	#/s	#/min	#/min
`fcr`	128/5	256/10	1024/0,6	2048/1,2

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.2: Využitie branch and bound metódy v priestore parametrov pre jednoparametrickú sústavu (Skalna1).



Obr. 7.7: `fcr` $\epsilon = 0{,}01$.



Obr. 7.8: `outerbound` $\epsilon_{ob} = 0{,}1$.

7.2 Dvojparametrická sústava

$$A(p) = \begin{pmatrix} p_1 & p_2 - 1 \\ p_2 & p_1 \end{pmatrix}, \quad b(p) = \begin{pmatrix} -p_2 + 1/3 \\ p_2 \end{pmatrix}, \quad p_1 \in [-2, -1], \quad p_2 \in [3, 5] \text{ [18]}.$$

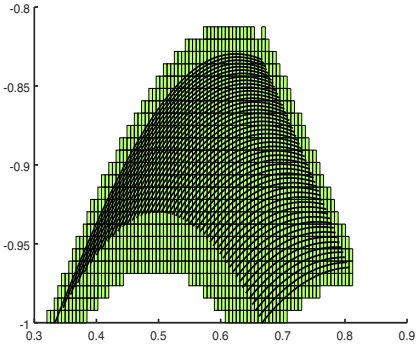
Hranice počiatočného boxu: $([-0{,}5;\ 1], [-1;\ -0{,}5])^{\mathrm{T}}$.

epsilon	**0,05**	**0,01**	**0,005**	**0,003**
	#/s	#/min	#/min	#/min
`intersect`	118/5	3039/2,3	11791/9	23363/18
`intersect2`	162/6	4698/3,1	18584/12	37064/24
`oettliPrager`	108/8	2962/3,2	11652/12	23108/25
`gaussSeidel`	467/13	7670/3,6	30647/14	118516/61
`hladiklp`	112/14	1210/3,8	3977/14	7347/34

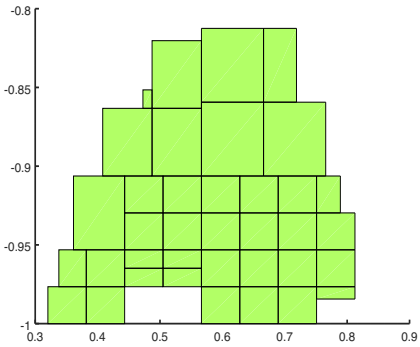
Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.3: Využitie branch and bound metódy v priestore riešení pre dvojparametrickú sústavu (Popova2).

Na nasledujúcích obrázkoch vidíme výslednú množinu z prerezávacej metódy `hladiklp` v rôznych podobách. Na obrázku 7.9 ju vidíme neupravenú spolu s riešením. Ďalej na 7.10 ju vidíme upravenú s funkciou `outerbound`.

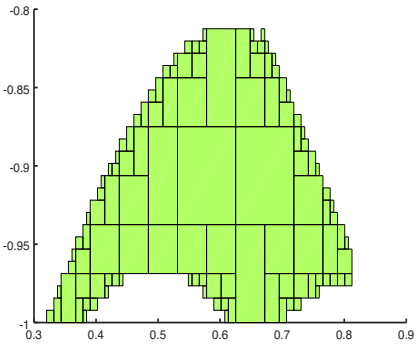


Obr. 7.9: `hladiklp` $\epsilon = 0{,}01$.

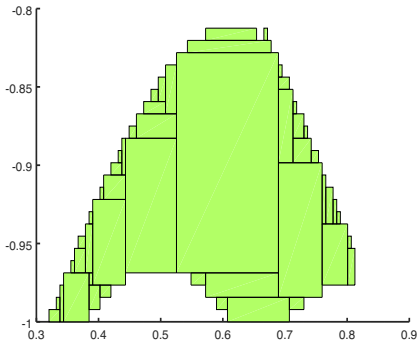


Obr. 7.10: `outerbound` $\epsilon_{ob} = 0{,}1$.

Na obrázkoch 7.11 a 7.12 ju vidíme upravenú so zlievajúcimi funkciami.



Obr. 7.11: `mergeboxes`.



Obr. 7.12: `mergeboxesheu`.

epsilon	**0,2**	**0,1**	**0,05**	**0,01**
	#/s	#/min	#/min	#/min
`fcr`	128/8	512/0,5	2048/1,8	32768/28

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.4: Využitie branch and bound metódy v priestore parametrov pre dvojparametrickú sústavu (Popova2).

7.3 Trojparametrická sústava

$$A(p) = \begin{pmatrix} 1 & p_1 \\ p_1 & p_2 \end{pmatrix}, \quad b(p) = \begin{pmatrix} p_3 \\ p_3 \end{pmatrix},$$
$$p_1 \in [0,1], \quad p_2 \in [-4,-1], \quad p_3 \in [0,2] \ [18].$$

Hranice počiatočného boxu: $([-2;\ 2{,}5], [-2{,}5;\ 2])^{\mathrm{T}}$.

epsilon	**0,5**	**0,1**	**0,05**	**0,03**
	#/s	#/min	#/min	#/min
`intersect`	108/6	1459/1,0	5725/3,6	22595/14
`intersect2`	156/7	2274/1,6	8984/5,9	35615/23
`oettliPrager`	103/9	1396/1,7	5557/6,1	22048/23
`gaussSeidel`	162/4	2819/1,1	11154/4,2	40894/16
`hladiklp`	91/9	729/1,7	2517/5,6	9164/23

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.5: Využitie branch and bound metódy v priestore riešení pre trojparametrickú sústavu (Popova3).

epsilon	**0,5**	**0,3**	**0,2**	**0,1**
	#/s	#/min	#/min	#/min
`fcr`	256/17	512/0,6	2048/1,8	16384/14

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.6: Využitie branch and bound metódy v priestore parametrov pre trojparametrickú sústavu (Popova3).

7.4 Štvorparametrická sústava

$$A(p) = \begin{pmatrix} 2p_1 & p_3 - p_1 \\ 2.5p_3 + p_2 & p_2 \end{pmatrix}, \quad b(p) = \begin{pmatrix} 2p_4 \\ 2p_4 \end{pmatrix},$$
$$p_1 \in [\frac{1}{2}, \frac{3}{2}], \quad p_2 \in [\frac{7}{10}, \frac{17}{10}], \quad p_3 \in [0, 1], \quad p_4 \in [\frac{13}{6}, \frac{17}{6}] \text{ [20].}$$

Hranice počiatočného boxu: $([-2;\ 6], [-3{,}5;\ 4])^{\mathrm{T}}$.

epsilon	**0,5**	**0,2**	**0,1**	**0,05**
	#/s	#/min	#/min	#/min
`intersect`	281/14	2120/1,6	8335/6,3	33028/25
`intersect2`	499/23	3935/2,9	15676/11,6	62563/47
`oettliPrager`	264/30	2053/2,9	8144/11,5	32447/46
`gaussSeidel`	244/6	2026/0,8	8082/3,4	32286/13
`hladiklp`	277/10	1935/1,1	7317/4,5	28338/18

Pozn: # − počet vrátených boxov, čas udávame v sekundách (s) alebo v minútach (min).

Tabuľka 7.7: Využitie branch and bound metódy v priestore riešení pre štvorparametrickú sústavu (Popova8).

7.5 Zhrnutie výsledkov

V priestore riešení máme uvedených päť podmienok ako je vidieť v tabuľkách, v ktorých sme porovnávali prerezávacie podmienky. Podmienka `fcr` vracala celý pôvodný počiatočný priestor, len rozsekaný na kúsky podľa daného ϵ. Tým pádom sa nám viac oplatí pôvodný priestor rozsekať na malé boxy podľa ϵ a teda nemusíme branchovať. Preto sme ju v priestore riešení netestovali.

Na základe porovnania v tabuľkách v priestore riešení odporúčame používať najmä prerezávaciu podmienku `hladiklp`. Výhoda tejto metódy sa prejavuje pri malých ϵ. Je výhodná už aj pri väčších ϵ, ale čím viac ho zmenšujeme, tým viac sa prejavuje výhoda tejto podmienky, dokonca rádovo v počte vrátených boxov. Čo sa týka času, tak je porovnateľná s ostatnými. Jedine v štvorparametrickej sústave sú s ňou porovnateľné aj ostatné podmienky prerezávania, kde dostávame veľmi vyrovnané výsledky. Tu je najrýchlejšia podmienka `gaussSeidel`, ktorá sa však v príkladoch s menším počtom parametrov ako štyri neprejavila výhodne. Všimli sme si, že podmienka `oettliPrager` vo väčšine prípadov nevracia množinu, ktorá obsahuje celé riešenie ako je vidieť na obrázku 7.3. Preto je v podstate nepoužiteľná.

Všeobecne výhoda prerezávacích podmienok v priestore riešení spočíva v tom, že pri väčšom počte parametrov v sústave sú lepšie ako prerezávacie podmienky v priestore parametrov, čo je vidieť aj v tabuľkách. V rámci testovania sme zistili, že v priestore parametrov nám funguje iba jedna prerezávacia podmienka a to `fcr`. Predpokladali sme, že aj podmienka `oettliPrager` bude výhodná, ale v skutočnosti sa správala rovnako ako v priestore riešení, že niekedy vracala množinu, ktorá neobsahovala celé riešenie. Preto sme ju netestovali a zaoberali sme sa len podmienkou `fcr`. Hlavnou výhodou tejto funkcie je, že pri malom počte parametrov je rádovo lepšia v porovnaní s podmienkami v priestore riešení a dokáže prerezávať aj cez veľmi malé ϵ, najmä pri jednoparametrickej sústave. Pri testovaní štvormarapetrickej sústavy sme ju ale nedokázali ani otestovať.

Pre sústavy s malým počtom parametrov odporúčame prerezávať v priestore parametrov a teda využiť podmienku fcr. Pri sústavách s viacerými parametrami odporúčame prerezávať v priestore riešení s podmienkou hladiklp.

Pre zaujímavosť sme si v niektorých prípadoch zobrazili množinu riešení. Dokonca sme ich aj optimalizovali s našimi navrhnutými algoritmami z kapitoly 5. Pre veľkú plochu je vhodné použiť metódu mergeboxesheu, pre úzke tvary zas metódu outerbound. Je však na užívateľovi, akú metódu preferuje.

8. Užívateľská dokumentácia

Popísané algoritmy sú implementované do intervalového balíku Lime, ktorý je vyvíjaný pre program Octave. Preto je každý algoritmus v tejto práci napísaný v tomto jazyku, navyše je voľne dostupný. Na intervalové počítanie využívame balík Interval package [6] ako nástroj Octave.

8.1 Lime

LIME (Library of Interval MEthods) je súbor nástrojov pre výpočty s intervalovými údajmi, ktorý vyvíja vedúci tejto práce RNDr. Jaroslav Horáček. Tento balík bude zverejnený v septembri tohoto roku.

8.2 Octave

Octave je voľne dostupný softvér [5] pre vykonávanie numerických výpočtov, ktorý poskytuje ďalšie možnosti numerického riešenia lineárnych a nelineárnych problémov a vykonávania numerických experimentov. Je veľmi podobný jazyku Matlab, takže väčšina programov je ľahko prenosná.

8.3 Interval package

Interval package je voľne dostupný súbor nástrojov pre Octave, ktorý je primárne navrhnutý na reálnu intervalovú aritmetiku. Umožňuje vyhodnotiť funkcie nad podmnožinami svojej domény. Výsledky sú overené, pretože intervalové výpočty automaticky sledujú chyby. Často sa využíva aj na aplikovanie rôznych dôkazov podporovaných počítačom alebo na programovanie s obmedzením. Niektoré platformy už majú tento balík automaticky zahrnutý v Octave. V každom prípade môže byť tento intervalový balík Interval package alternatívne nainštalovaný pomocou príkazu *pkg* z príkazového riadku priamo v Octave. Najnovšiu verziu potom môžeme stiahnuť s príkazom `-forge` [17]. Celá inštalácia v príkazovom riadku Octave vyzerá takto:

```
>> pkg install -forge interval
```

Následne ju načítame v príkazovom riadku Octave:

```
>> pkg load interval
```

8.4 Funkcie

V tejto časti kapitoly opíšeme základné funkcie, ktorými môžeme vyskúšať algoritmy z tejto práce. Pri každej si uvedieme krátky popis, vstupné a výstupné parametre.

8.4.1 tests

Funkcia `tests` pomáha k ľahšiemu načítaniu parametrických intervalových lineárnych sústav. Môžeme si ju predstaviť ako databázu príkladov. Vstupný parameter sa zadáva ako reťazec ohraničený úvodzovkami ako napríklad `"Popova1"`.

Syntax: `[initx, pk, Ak, bk, A, b] = tests(name)`

Vstupný parameter:

name	meno jednotlivých príkladov parametrických intervalových lineárnych sústav, zadáva sa ako reťazec ohraničený úvodzovkami ako napríklad `"Popova1"`

Výstupné parametre:

initx	vektory stĺpcového intervalu / matica intervalových stĺpcových vektorov
pk	stĺpcový vektor intervalov - hranice parametrov
Ak	bunkové pole matíc
bk	bunkové pole stĺpcových vektorov
A	reálna matica
b	reálny vektor stĺpca

8.4.2 pilsbabonx

Funkcia aplikuje branch and bound metódu na parametrické intervalové lineárne sústavy $\boldsymbol{A}x = \boldsymbol{b}$ (na priestor riešení). Využíva iteračné metódy prerezávaní pre klasické intervalové sústavy.

Syntax: `[solution, disc, depth] = pilsbabonx(initx, eps, pk, Ak, bk, A, b, typeofprune)`

Vstupné parametre:

initx	vektory stĺpcového intervalu / matica intervalových stĺpcových vektorov
eps	minimálna šírka intervalu, ktorý je povolený na rozdelenie
pk	stĺpcový vektor intervalov - hranice parametrov
Ak	bunkové pole matíc
bk	bunkové pole stĺpcových vektorov
A	reálna matica
b	reálny vektor stĺpca
typeofprune	názov prerezávacej metódy, ktorú chceme využiť, zadávajú sa ako reťazce v úvodzovkách (`"intersect"`, `"intersect2"`, `"gaussSeidel"`, `"oettliPrager"`, `"hladiklp"`, `"fcr"`)

Výstupné parametre:

solution	boxy, ktoré môžu obsahovať riešenie
disc	boxy, ktoré určite neobsahujú riešenie
depth	určuje hĺbku, v ktorej je box zahodený

8.4.3 pilsbabonparams

Funkcia aplikuje branch and bound metódu na parametrické intervalové lineárne sústavy $\boldsymbol{A}(p)x = \boldsymbol{b}(p)$ (na priestor parametrov). Využíva iteračné metódy prerezávaní pre klasické intervalové sústavy.

Syntax: `[solution] = pilsbabonparams(initx, eps, pk, Ak, bk, A, b, typeofprune)`

Vstupné parametre:

initx	vektory stĺpcového intervalu / matica intervalových stĺpcových vektorov
eps	minimálna šírka intervalu, ktorý je povolený na rozdelenie
pk	stĺpcový vektor intervalov - hranice parametrov
Ak	bunkové pole atíc
bk	bunkové pole stĺpcových vektorov
A	reálna matica
b	reálny vektor stĺpca
typeofprune	názov prerezávacej metódy, ktorú chceme využiť, zadávajú sa ako reťazce v úvodzovkách (`"oettliPrager"`, `"fcr"`)

Výstupný parameter:

solution	boxy, ktoré môžu obsahovať riešenie

8.4.4 mergeboxes

V branch and bound metóde používame rozdelenie podľa väčšej strany, táto funkcia spája malé boxy (synov) s ich predchádzajúcimi boxmi (rodičmi). Ak sú všetci označení, tak je to možné riešenie.

Syntax: `[newboxes] = mergeboxes(boxes, initx)`

Vstupné parametre:

boxes	boxy získané z branch and bound metódy
initx	počiatočný ohraničujúci box pred branch and bound metódou

Výstupný parameter:

newboxes	výstup nových zlúčených boxov

8.4.5 mergeboxesheu

V branch and bound metóde používame rozdelenie podľa väčšej strany, táto funkcia spája získané boxy využitím heuristiky.

Syntax: `[newboxes] = mergeboxesheu(boxes, initx)`

Vstupné parametre:

boxes	boxy získané z branch and bound metódy
initx	počiatočný ohraničujúci box pred branch and bound metódou

Výstupný parameter:

newboxes	výstup nových zlúčených boxov

8.4.6 premerge

Prechodná funkcia medzi výstupom branch and bound metódy cez priestor parametrov a vizualizáciou mergeboxesheu. Nafúkne boxy, aby ich bolo možné zaradiť do bitmapovej mriežky.

Syntax: `[newboxes] = premerge(boxes, initx)`

Vstupné parametre:

boxes	boxy získané z branch and bound metódy
initx	počiatočný ohraničujúci box pred branch and bound metódou

Výstupný parameter:

newboxes	výstup nových nafúknutých boxov

8.4.7 outerbound

Ohraničuje boxy získané branch and bound metódou rekurzívne podľa kvadrantov z vonka, pokiaľ veľkosť kvadrantu vyhovuje obmedzujúcej podmienke.

Syntax: `[newboxes] = outerbound(boxes, limitbound)`

Vstupné parametre:

boxes	boxy získané z branch and bound metódy
limitbound	obmedzujúca podmienka pre rekurziu

Výstupný parameter:

newboxes	výstup nových boxov

8.4.8 plotboxes

Základná funkcia pre vizualizáciu získaných boxov. Na výstup priamo vykresľuje dané boxy s výberom farby. Farby môžu byť špecifikované ako RGB trojice s hodnotami v rozmedzí od nuly do jednotky ([0-1 0-1 0-1]) alebo podľa názvu. Uznané názvy farieb sú: "blue"(modrá), "black"(čierna), "cyan"(azúrová), "green"(zelená), "magenta"(purpurová), "red"(červená), "white"(biela) a "yellow" (žltá).

Syntax: `plotboxes(boxes, color)`

Vstupné parametre:

boxes	boxy, ktoré chceme zobraziť
color	je voliteľný parameter na výber farby, ktorý sa zadáva v prípade názvu do úvodzoviek alebo v prípade RGB do hranatých zátvoriek, preddefinovaná je svetlozelená

8.4.9 plotdiscardboxes

Funkcia zobrazuje boxy a zafarbuje ich podľa hĺbky zahodenia. Na výstup priamo vykresľuje dané boxy. Čím bol box neskôr vyradený, že neobsahuje riešenie, tým dostane svetlejšiu farbu a naopak, prvé vyradené boxy budú tmavšie.

Syntax: plotdiscardboxes(boxes, depth)

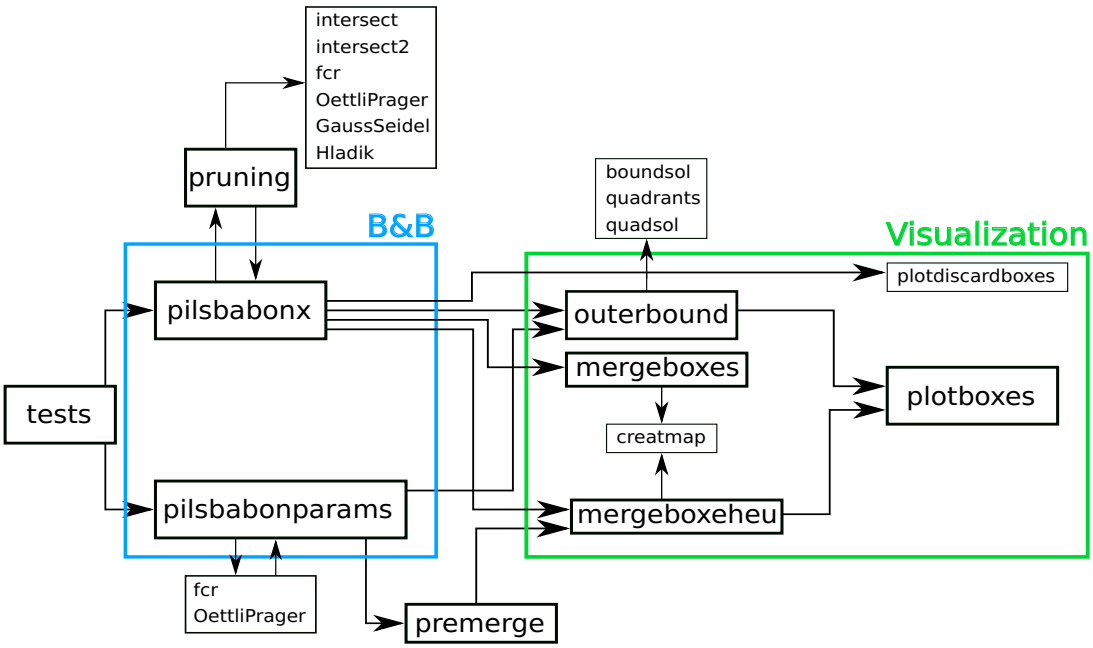
Vstupné parametre:

boxes	boxy, ktoré chceme zobraziť
depth	určuje hĺbku, v ktorej je box orezaný/zahodený

9. Programátorská dokumentácia

9.1 Štruktúra funkcií

V našom kóde nenachádzame zložité štruktúry. Skladá sa zo samostatných funkcií a preto programová časť tejto práce nie je rozsiahla. V nasledujúcich sekciách si ukážeme celkovú štruktúru a oboznámime sa s úlohami jednotlivých funkcií. Uvedieme si aj pomocné funkcie, využívané z balíku Lime. Jednotlivé vstupné a výstupné parametre nebudeme uvádzať, pretože už boli zmienené v kapitole 8.



Obr. 9.1: Mapa základných funkcií.

9.2 Funkcie

Funkcia `tests` obsahuje sadu parametrických intervalových lineárnych sústav s rozličným počtom parametrov. Pri každom z nich sa v komentári odkazujeme na zdroj a na špecifikáciu daného príkladu pre dodatočné vyhľadanie v použitej literatúre. Názvy jednotlivých sústav sú odvodené od mien autorov príslušnej literatúry.

Funkcia `pilsbabonx` implementuje branch and bound metódu na priestor riešení. Ukladanie výsledných boxíkov prebieha za pomoci zásobníku kvôli rýchlosti algoritmu. Je to výhodnejšie ako ich vkladanie do poľa. Táto funkcia má ako jeden zo vstupných parametrov typ prerezávacej podmienky, ktorú má použiť, pričom samotné delenie boxov nastáva v polovici dlhšej strany. Jednotlivo si tieto podmienky uvedieme neskôr v tejto kapitole. Presnejší popis fungovania algoritmu nájdeme v časti 3.

Podobne ako predchádzajúca funkcia `pilsbabonx`, funkcia `pilsbabonparams` implementuje metódu branch and bound s tým rozdielom, že prerezáva priestor parametrov a nie priestor riešení. Rovnako pracuje aj so zásobníkom. Popis tejto funkcie nájdeme v kapitole 6.

Funkcie začínajúce sa na `prune_` sú implementácie jednotlivých prerezávacích podmienok.

Funkcia `prune_intersect` implementuje priamo relaxačnú podmienku 4.1.

Funkcia `prune_intersect2` implementuje reziduálny prístup 4.3.

Funkcia `prune_fcr` implementuje podmienku využívajúcu hodnosť matice 4.2. Na jej implementáciu si z balíku Lime požičiavame niekoľko funkcií. Funkciu `isunsfcr` na overenie, či intervalová sústava má prázdne riešenie. Kontroluje plnú hodnosť matice a ak ju matica má, tak sa overí. Funkciu `ifcr`, ktorá rozhoduje, či matica má plnú hodnosť. Funkciu `imaxnorm`, ktorá počíta maximovú normu pre intervalovú maticu.

Funkcia `prune_oettli` implementuje podmienku využívajúcu vetu Oettli-Prager 4.4.

Funkcia `prune_gs` implementuje Gauss-Seidelovu metódu 4.5. V implementácii využívame pomocnú funkciu `ilsgsstep`, ktorá je súčasťou balíku Lime a zabezpečuje jeden krok Gauss-Seidelovej iterácie.

Funkcia `prune_hladiklp` implementuje podmienku využívajúcu vetu, ktorú nám poskytol doc. Hladík 4.6. V implementácii využívame lineárne programovanie. Na jej riešenie je v Octave vopred definovaná funkcia `glpk`.

Spracovanie získaných boxov riešime hneď s niekoľkými funkciami.

Funkcia `outerbound` je implementáciou štvrtinovej metódy z kapitoly 5.1. Využívame v nej pomocné funkcie `quadrants` na rozdelenie množiny na 4 skupiny. Funkciu `quadsol` používame na rozdelenie boxov do štyroch skupín a funkciu `boundsol` na získanie intervalového obalu jednotlivých skupín.

Funkcia `mergeboxes` implementuje zlievanie boxov, ktorej návrh sme popísali v podkapitole 5.2. Na vytvorenie bitmapy využívame pomocnú funkciu `creatmap`, ktorá ju následne aj vyplní s 0/1. Rekurzívne zlievanie jednotlivých boxov riešime pomocou zásobníku.

Funkcia `mergeboxesheu` implementuje návrh na zlievanie boxíkov z podkapitoly 5.3. Využíva funkciu `creatmap` podobne ako funkcia `mergeboxes`. Priamo v nej je implementovaná funkcia `spread` na zlučovanie jednotlivých boxov podľa popísanej heuristiky.

Funkcia `premerge` slúži na nafúknutie jednotlivých boxov. Je určená ako prechodová funkcia medzi získanými boxmi z funkcie `pilsbabonparams` a zlievajúcou funkciou `mergeboxesheu`. Dôvod je popísaný v kapitole 6.

Na konkrétne zobrazenie jednotlivých boxov, prípadne množiny riešení je určená funkcia `plotboxes`. Na vizualizáciu postupu prerezávania podľa hĺbky je implementovaná funkcia `plotdiscardboxes`.

9.3 Funkcie z Lime

Uvedieme si ešte funkcie, ktoré využívame z balíku Lime, okrem už spomenutých. Funkciu `plotilssol` využívame na zobrazovanie množiny riešení pre intervalové lineárne sústavy. Funkciu `plotpilssol` zas na zobrazovanie riešení parametrických intervalových lineárnych sústav. Tieto funkcie nie sú súčasťou odovzdanej prílohy, pretože ich nepotrebujeme nevyhnutne k našej práci a sú súčasťou balíku Lime.

Záver

V tejto práci sme si vysvetlili rozdiely medzi jednotlivými lineárnymi sústavami, tj. medzi intervalovými lineárnymi sústavami a parametrickými intervalovými lineárnymi sústavami. Popísali sme základnú metódu branch and bound v podobe, v akej sme ju využívali. Ďalej sme navrhli rôzne prerezávacie podmienky, ktoré sme testovali s použitím branch and bound metódy na parametrických sústavách s rôznym počtom parametrov. Testovanie prebiehalo na priestore riešení ako aj na priestore parametrov. Zistili sme, že pre sústavy s malým počtom parametrov je najvýhodnejšie prerezávať v priestore parametrov s využitím podmienky `fcr` a pre sústavy s viacerými parametrami je vhodnejšie prerezávanie v priestore riešení s podmienkou `hladiklp`.

Pre získané boxy sme navrhli tri možnosti pre spracovanie. Ich jednotlivé využitie záleží na užívateľovi, keďže každá má výhodu v niečom inom. Jedna je výhodná, pretože sa snaží maximalizovať veľkosť prvotného boxu, teda nám tam vznikne aspoň jeden väčší box. Výhoda druhej je, že pomocou vstupného parametru vieme obmedziť šírku výsledných boxov a tretej zas, že výsledné boxy zarovnáva do jednej úrovne. Zaujímavé výsledky môžu vzniknúť aj ich kombináciou. Všetky uvedené metódy sme implementovali v intervalovom balíku Lime pre Octave. Tento balík obsahuje metódy pre prácu s intervalovými maticami ako aj metódy na vizualizáciu parametrických a intervalových lineárnych sústav a mnoho ďalších metód ohľadne intervalových matíc.

Teoretická časť tejto práce môže slúžiť aj ako stručný úvod do riešenia parametrických intervalových lineárnych sústav. Ďalej môže byť určitou inšpiráciou pre ďalšiu prácu s prerezávacími podmienkami. Bolo by zaujímavé ich navzájom vhodne kombinovať, napríklad, že prvotné prerezanie by prerezalo počiatočnú množinu rýchlo a potom by sa použila nejaká efektívnejšia podmienka prerezávania. Ďalej by sa dali aj navzájom kombinovať prerezávacie podmienky v priestore riešení a v priestore parametrov.

Zoznam použitej literatúry

[1] Bader, D. A., Hart, W. E. a Phillips, C. A. (2005). Parallel algorithm design for branch and bound. In *Tutorials on Emerging Methodologies and Applications in Operations Research*, pages 5–1. Springer.

[2] Beeck, H. (1973). Charakterisierung der lösungsmenge von intervallgleichungssystemen. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, **53**(12), T181–T182.

[3] Clausen, J. (1999). Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30.

[4] Garajová, E. (2014). Intervalový solver nelineárních podmínek. *Bakalářská práce, Matematicko-fyzikální fakulta University Karlovy v Praze.*

[5] Gnu.org. (2018). Gnu octave [online]. URL `https://www.gnu.org/software/octave`.

[6] Heimlich, O. (2016). Interval arithmetic in gnu octave. *SWIM 2016.*

[7] Hladík, M. (2012). Enclosures for the solution set of parametric interval linear systems. *International Journal of Applied Mathematics and Computer Science*, **22**(3), 561–574.

[8] Horáček, J. (2011). Přeurčené soustavy intervalových lineárních rovnic. *Diplomová práce, Matematicko-fyzikální fakulta University Karlovy v Praze.*

[9] Jansson, C. (1991). Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side. *Computing*, **46** (3), 265–274.

[10] Král, O. (2016). Intervalové lineární soustavy rovnic s lineárními závislostmi. *Bakalářská práce, Matematicko-fyzikální fakulta University Karlovy v Praze.*

[11] Kulpa, Z., Pownuk, A. a Skalna, I. (1998). Analysis of linear mechanical structures with uncertainties by means of interval methods. *Computer Assisted Mechanics and Engineering Sciences*, **5**(4), 443–477.

[12] Land, A. H. a Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520.

[13] Moore, R. E., Kearfott, R. B. a Cloud, M. J. (2009). *Introduction to interval analysis.* SIAM.

[14] Nazari, V. a Notash, L. (2014). Parametric method for motion analysis of manipulators with uncertainty in kinematic parameters. In *Advances on Theory and Practice of Robots and Manipulators*, pages 9–17. Springer.

[15] Neumaier, A. (1990). *Interval methods for systems of equations*, volume 37. Cambridge university press.

[16] Neumaier, A. a Pownuk, A. (2007). Linear systems with large uncertainties, with applications to truss structures. *Reliable Computing*, **13**(2), 149–172.

[17] Octave.sourceforge.io. (2018). Gnu octave interval package manual: Getting started. [online]. URL `https://octave.sourceforge.io/interval/package_doc/Getting-Started.html#Getting-Started`.

[18] Popova, E. a Krämer, W. (2008). Visualizing parametric solution sets. *BIT Numerical Mathematics*, **48**(1), 95–115.

[19] Popova, E. D. (2009). Explicit characterization of a class of parametric solution sets. *Comptes rendus de l'Académie bulgare des Sciences*, **62**(10), 1207–1216.

[20] Popova, E. D. (2012). Explicit description of ae solution sets for parametric linear systems. *SIAM Journal on Matrix Analysis and Applications*, **33**(4), 1172–1189.

[21] Rohn, J. (2005). A handbook of results on interval linear problems.

[22] Rohn, J. (2006). Solvability of systems of interval linear equations and inequalities. In *Linear optimization problems with inexact data*, pages 35–77. Springer.

[23] Rohn, J. (2014). Verification of linear (in) dependence in finite precision arithmetic. *Mathematics in Computer Science*, **8**(3-4), 323–328.

[24] Rump, S. M. (1994). Verification methods for dense and sparse systems of equations. In *J. Herzberger, ed., Topics in validated computations*, pages 63–136.

Zoznam obrázkov

Zoznam tabuliek